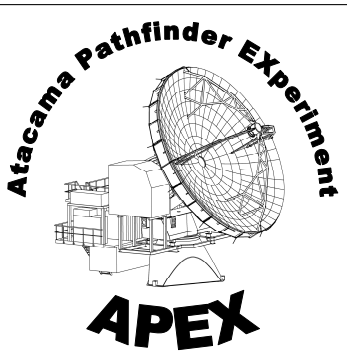


**Atacama Pathfinder EXperiment**

User Manual

| APEX-MPI-MAN-0011 |
|---|
| |
| Revision: 3.1 |
| |
| Release: March 3, 2015 |
| |
| Category: 4 |
| |
| Author: Dirk Muders |

# APECS User Manual

Dirk Muders

Keywords: APECS, Observing, Operating

| | |
|---|---|
| Author Signature: Dirk Muders | Date: March 3, 2015 |
| Approved by: T. Klein<br>Institute: APEX | Signature: T. Klein<br>Date: March 3, 2015 |
| Released by: T. Klein<br>Institute: APEX | Signature: T. Klein<br>Date: March 3, 2015 |

*Change Record*

| Revision | Date | Author | Section/ Page affected | Remarks |
|---|---|---|---|---|
| 1.0 | 2005-08-26 | Dirk Muders | All | Initial Version |
| 1.1 | 2006-06-16 | Dirk Muders | Selected paragraphs | Updated to APECS 0.9. |
| 1.11 | 2006-06-22 | Dirk Muders | 14 | Updated FLASH frequency ranges. |
| | | | 18 | Updated "calibrate" command syntax. |
| 1.12 | 2006-09-03 | Dirk Muders | Selected paragraphs | Updated to APECS 1.0: |
| | | | | Added spiral mode setup commands. |
| | | | | Updated "use_ref" documentation. |
| | | | | Fixed "derotate" command description. |
| 1.13 | 2007-04-03 | Dirk Muders | Introduction | Updated project directory structure. |
| | | | apecs commands | Added "drift" command. Iterated "project_id" and "spiral_setup" commands. |
| | | | | Updated instrument list. |
| | | | CORBA CDB reference | Updated CORBA component list. |
| 1.14 | 2007-06-08 | Dirk Muders | apecs commands | Updated "Heterodyne frontend name", "source", "raster" and "otf" command syntax. |
| | | | | Added "hexa" command. |
| 1.15 | 2007-10-17 | Dirk Muders | apecs commands | Updated to APECS 1.1. |
| | | | | Updated descriptions of commands. |
| | | | | Added new commands. |
| 2.0-2.2 | 2010-03-25 | Dirk Muders | New apecs commands | Updated to APECS 2.2. |
| | | | Observing / Operating | New server, observing, and network setup |
| | | | Instruments | Updated list of APEX frontends and backends. |
| | | | CORBA CDB reference | Updated CORBA component list. |
| 2.2.1 | 2010-09-21 | Dirk Muders | apecs commands | Added `remote_control` command. |
| | | | | Updated `project_id` description. |
| | | | | Updated `frontends` and `backends` description. |
| | | | | Added new instruments. |
| | | | All pages | Fixed typos. |
| 2.3-2.5 | 2012-04-11 | Dirk Muders | All pages | Updated to APECS 2.3 / ACS 9.0 / SL 5.5 |
| | | | | Updated to APECS 2.4 / ACU 8200. |
| | | | | Updated to APECS 2.5 / ACS 10.1. |
| | | | | New instruments. |
| | | | | New APECS server setup. |
| 2.5.1-2.6 | 2013-04-29 | Dirk Muders | apecs commands | Updated to APECS 2.6 / ACS 11.0 |
| | | | | Updated command descriptions. |
| | | | | Updated `frontends` and `backends` description. |
| | | | CORBA CDB reference | Updated CORBA component list. |
| 2.7 | 2014-02-28 | Dirk Muders | Map setup; instruments | Updated to APECS 2.7 / ACS 12.1. |
| | | | CORBA CDB reference | Updated CORBA component list. |
| 3.0 | 2014-07-21 | Dirk Muders | What's new section | Updated to APECS 3.0 / ACS 12.3 / SL 6.5 / Gildas June 2014. |
| | | | CORBA CDB reference | Updated CORBA component list. |
| 3.1 | 2015-02-03 | Dirk Muders | What's new section | Updated to APECS 3.1 / ACS 2014.6 / SL 6.6 / Gildas July 2014. |
| | | | CORBA CDB reference | Updated CORBA component list. |
| | | | apecs commands | Update for `calibrate` command. |

# Contents


**1 What's new ? 8**
1.1 Revision 3.1 . . . 8
1.2 Revision 3.0 . . . 8
1.3 Revision 2.7 . . . 8
1.4 Revision 2.6 . . . 8
1.5 Revision 2.5.1 . . . 8
1.6 Revision 2.5 . . . 8
1.7 Revision 2.4 . . . 9
1.8 Revision 2.3 . . . 9
1.9 Revision 2.2.1 . . . 9
1.10 Revision 2.2 . . . 9
1.11 Revision 2.1 . . . 10
1.12 Revision 2.0 . . . 10
1.13 Revision 1.15 . . . 10
1.14 Revision 1.14 . . . 10
1.15 Revision 1.13 . . . 10
1.16 Revision 1.12 . . . 10

**2 Overview 11**

**3 Observing with the APEX Telescope 13**
3.1 Introduction . . . 13
3.2 apecs commands . . . 15
3.2.1 General . . . 15
3.2.1.1 `project_id` . . . 15
3.2.1.2 `operator_id` . . . 16
3.2.1.3 `observer_id` . . . 16
3.2.1.4 `show` . . . 16
3.2.1.5 `load` . . . 16
3.2.1.6 `go` . . . 16
3.2.1.7 `track` . . . 16
3.2.1.8 `cancel` . . . 16
3.2.1.9 `exec_apecs_script` . . . 16
3.2.1.10 `save_history` . . . 17
3.2.1.11 `save_defaults` . . . 17
3.2.1.12 `load_defaults` . . . 17
3.2.1.13 `reset_defaults` . . . 17
3.2.1.14 `observe` . . . 17
3.2.1.15 `continuous_data` . . . 17
3.2.1.16 `skip_hardware_setup` . . . 17
3.2.1.17 `remote_control` . . . 18
3.2.2 Catalogs . . . 18
3.2.2.1 `sourcecats` . . . 18
3.2.2.2 `ephemerides` . . . 18

3.2.2.3 linecats . . . 18
3.2.3 Instruments . . . 19
3.2.3.1 frontends . . . 21
3.2.3.2 <Frontend name>.feeds . . . 21
3.2.3.3 <Continuum frontend name>.configure . . . 21
3.2.3.4 <Heterodyne frontend name>.configure . . . 22
3.2.3.5 <Heterodyne frontend name>.line . . . 23
3.2.3.6 <Heterodyne frontend name>.derotate . . . 23
3.2.3.7 <Frontend name>.backends . . . 23
3.2.3.8 <Continuum backend name>.configure . . . 24
3.2.3.9 <Continuum backend name>_group.configure . . . 24
3.2.3.10 <Spectral backend name>.configure . . . 24
3.2.3.11 <Spectral backend name>_group.configure . . . 24
3.2.4 Target . . . 24
3.2.4.1 source . . . 25
3.2.5 Calibration . . . 25
3.2.5.1 calibrate . . . 25
3.2.5.2 skydip . . . 25
3.2.5.3 point . . . 26
3.2.5.4 pcorr . . . 26
3.2.5.5 pcorr_reset . . . 26
3.2.5.6 focus . . . 26
3.2.5.7 fcorr . . . 27
3.2.5.8 fcorr_reset . . . 27
3.2.5.9 use_foc_temp_corr . . . 27
3.2.5.10 set_tilts . . . 27
3.2.5.11 reset_tilts . . . 27
3.2.5.12 use_linear_sensors . . . 27
3.2.5.13 use_tiltmeters . . . 27
3.2.5.14 set_cold_params . . . 28
3.2.5.15 reset_cold_params . . . 28
3.2.6 Observing Patterns . . . 28
3.2.6.1 offset . . . 28
3.2.6.2 reference . . . 28
3.2.6.3 use_ref . . . 29
3.2.6.4 on . . . 30
3.2.6.5 raster . . . 30
3.2.6.6 hexa . . . 31
3.2.6.7 hexa25 . . . 31
3.2.6.8 otf . . . 32
3.2.6.9 drift . . . 32
3.2.6.10 repeat . . . 33
3.2.7 Stroke Mode . . . 33
3.2.7.1 linear . . . 33
3.2.7.2 spiral . . . 33
3.2.7.3 lissajous . . . 33
3.2.8 Switch Mode . . . 34
3.2.8.1 tp . . . 34
3.2.8.2 wob . . . 34
3.2.8.3 fsw . . . 34
3.2.9 Antenna . . . 34
3.2.9.1 tolerance . . . 34
3.2.9.2 park . . . 34
3.2.9.3 zenith . . . 35
3.2.9.4 stow . . . 35
3.2.9.5 unstow . . . 35

3.2.9.6 stow_wobbler . . . 35
3.2.9.7 init_wobbler . . . 35
3.2.9.8 reset_wobbler . . . 35
3.2.9.9 switch_c_optics . . . 35
3.3 A typical Observing Session . . . 36
3.3.1 Source Setup . . . 36
3.3.2 Continuum Instrument Setup . . . 36
3.3.3 Spectral Line Instrument Setup . . . 36
3.3.4 Initial Calibrations . . . 36
3.3.5 Continuum Observations . . . 37
3.3.6 Spectral Line Observations . . . 37
3.4 Macros and Loops . . . 37
3.5 User defined Commands . . . 38
3.6 Notes and Caveats . . . 38

**4 Operating the APEX Telescope 39**
4.1 Introduction . . . 39
4.2 Starting the APECS servers . . . 39
4.3 Configurations . . . 40
4.4 Troubleshooting . . . 41
4.5 Notes and Caveats . . . 44

**5 CORBA CDB Reference 45**

# List of Figures

2.1 APECS Pipeline Structure . . . . . 11

3.1 Rotated coordinate systems in APECS. . . . . 29

4.1 APECS deployment at the site in Chile. . . . . 40

# List of Tables


3.1 APECS logging GUI commands. . . . 14
3.2 Example of an APEX source catalog. . . . 18
3.3 Example of an APEX line catalog. . . . 19
3.4 APEX heterodyne and continuum frontends. The receiver cabins are named A (left Nasmyth), B (right Nasmyth) and C (Cassegrain). . . . 19
3.5 APEX spectral and continuum backends . . . . 20
3.6 Possible FEBE combinations between the APEX heterodyne frontends and the spectral backends. 21
3.7 Possible FEBE combinations between the APEX heterodyne frontends and the continuum backends. . . . 22
3.8 Possible FEBE combinations for the APEX continuum frontends. . . . 22

4.1 APECS hosts and standard processes. . . . 42
4.2 Known APECS problems and their solutions. . . . 43
4.3 Core APECS application restart commands. . . . 44

5.1 APECS CORBA components, containers, server hosts and processes providing the actual implementation. . . . 45
5.2 APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.). . . . 46
5.3 APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.). . . . 47
5.4 APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.). . . . 48
5.5 APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.). . . . 49
5.6 APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.). . . . 50

# Chapter 1

# What's new ?

## 1.1 Revision 3.1

In January 2015 APECS was upgraded to revision 3.1. This version includes a small upgrade of the operating system to Scientific Linux 6.6. ACS was updated to the latest version used at ALMA (2014.6) plus additional upgrades of Python to version 2.7.8 and scipy to 0.14.0. The Gildas software was updated to version jul14c, the latest available release with patches for enhanced zero spacing processing for ALMA with data from other observatories such as APEX.
The `calibrate` command (see 3.2.5.1) was extended to accept a list of times to be able to set up different times for the SKY, HOT and COLD subscans.

## 1.2 Revision 3.0

In June 2014 APECS was upgraded to revision 3.0. The major change was the switch to an updated Linux distribution (Scientific Linux 6.5) and the accompanying ACS version (12.3). The new Linux allowed to upgrade the Gildas version to June 2014. The updated OS makes APECS usable for more recent server hardware and new libraries facilitate improving functionality in further releases.

## 1.3 Revision 2.7

In February 2014 APECS was upgraded to ACS 12.1. The major change in ACS is the switch to Python 2.7 and scipy 0.13. The APECS map angle setup is explained in more detail. The list of instruments has been updated.

## 1.4 Revision 2.6

In March 2013 APECS was upgraded to ACS 11.0. The `apecs` commands descriptions and the tables of instruments and CORBA components are updated.

## 1.5 Revision 2.5.1

Switched to ACE/TAO 5.8.1 after fixing the ABM crash problems seen in APECS 2.5.

## 1.6 Revision 2.5

In January 2012 APECS was upgraded to ACS 10.1. This maintenance upgrade was installed to follow the ALMA ACS developments and to benefit from improvements and bug fixes.
Some of the APECS servers have been replaced with new machines. The deployment diagram and tables have been updated accordingly.

The apecs commands chapter has been updated to document a number of small changes introduced in 2011.

## 1.7 Revision 2.4

In February 2011 the Antenna Control Unit (ACU) was upgraded to model ACU 8200 involving an ICD change between the APECS ABM and the ACU. The new ICD was implemented in APECS 2.4. The normal APECS functionality was not changed compared to APECS 2.3.

## 1.8 Revision 2.3

In January 2011 APECS was upgraded to ACS 9.0 and Scientific Linux 5.5. This maintenance upgrade did not change the functionality compared to APECS 2.2.1 that was used until the end of 2010. APECS 2.3 was installed as a fallback system still using the the Antenna Control Unit ACU 8100.

## 1.9 Revision 2.2.1

Added remote_control command. Updated project_id description to document new parameter values. The frontends and backends commands now also take names without quotes and brackets. Updated the instrument and CDB tables.

## 1.10 Revision 2.2

The revision describes the APECS 2.2 release version as of March 2010. The main change is the switch to the current version of the ALMA Common Software (ACS v8.1) and an updated Linux operating system (Scientific Linux 5.3). APECS 2.2 was installed in January 2010.
In addition this version of the APECS manual describes some important changes and additions regarding the "apecs" commands:

- New logging GUI commands
- New load and track commands
- Frontend / backend / backend section group "configure" commands
- Lissajous setup
- Position angle for the "raster" mode (NB: the addition of this option changed the "raster" command interface)
- Version controlled observing and reduction scripts

Some old commands have been deprecated and replaced by new ones. The old commands will continue to work for some time until they will be removed. The following table gives an overview.

| Old command | New command |
|---|---|
| <Continuum frontend name> | <Continuum frontend name>.configure |
| <Heterodyne frontend name> | <Heterodyne frontend name>.configure |
| <Continuum backend name> | <Continuum backend name>.configure |
| <Continuum backend name>_group | <Continuum backend name>_group.configure |
| <Spectral backend name> | <Spectral backend name>.configure |
| <Spectral backend name>_group | <Spectral backend name>_group.configure |
| spiral_setup | spiral |
| use_spiral | Obsolete. Automatically set by new spiral command |
| run_macro | execfile, exec_apecs_script |

With APECS 2.2 we begin providing version controlled observing and data reduction scripts in the `$APECSROOT/share/apecs/`, `$APECSROOT/share/gildas/` and `$APECSROOT/share/boa/` directories. The observing scripts can be easily used via the new `exec_apecs_script` command which automatically prepends the corresponding path. One can also run them via "`execfile(APECS_SCRIPTS+'<script name>')`".
The list of existing frontends and backends and the tables of CORBA component names has been updated. The ACS alarm system has been enabled for BACI properties to provide alarms for critical monitor points such as rack or cabin temperatures.

## 1.11 Revision 2.1

This revision of APECS was introduced in mid 2009 to fix a bug in VxWorks that caused the ABM to crash quite often. The VxWorks operating system was upgraded from version 6.6 to 6.7.

## 1.12 Revision 2.0

Revision 2.0 of APECS was installed in January 2009. It was a major change in several aspects. The version of the ALMA Common Software (ACS) was upgraded from 2.0.1 to 8.0, the Linux version changed from RedHat 7.2 to Scientific Linux 5.2 and the server and network hardware was upgraded to modern machines and gigabit bandwidths to be able to handle the high data rates of the current and future instrumentation at APEX. Using an up-to-date ACS allows to benefit from the ongoing software developments of the ALMA project and to use new ACS features.

## 1.13 Revision 1.15

This revision describes the APECS 1.1 release version as of October 17$^{\text{th}}$, 2007. The changes are the iteration of the "frontends", "<Heterodyne frontend name>.line", "raster", "hexa", "otf", "on", "source", "offset", "reference", "stow" and "unstow" command syntax and descriptions. Added "set_cold_params", "reset_cold_params", "stow_wobbler", "reset_wobbler", "init_wobbler", "use_focus_compensation", "zenith" and "switch_c_optics" commands.
Note the important change that the "frontends" command now reloads all pointing models from file and issues warnings if the models have changed.

## 1.14 Revision 1.14

Addition of the "hexa" command and updates of the "source", "raster" and "otf" command syntax.

## 1.15 Revision 1.13

This minor revision includes the description of the "drift" command which is mainly used by the SZ project. The "project_id" and "spiral_setup" commands have been iterated. In addition, the project directory structure and the list of components have been updated to reflect the current status.

## 1.16 Revision 1.12

This revision describes APECS 1.0 as of September $3^{rd}$, 2006. The main changes are the introduction of the spiral observing modes for bolometer observations and changing the mode parameter of the `use_ref` command to accept values 'on' and 'off' rather than 1 and 0 to homogenize the commands for various flag settings withing `apecs`.

# Chapter 2

# Overview

The Atacama Pathfinder EXperiment (APEX) Telescope is controlled by the "APEX Control System" (APECS). APECS is based on the "ALMA Common Software" (ACS) and the "(ALMA) Test Interferometer Control Software" (TICS). ACS provides the CORBA-based middleware communication layer to interface the hardware components to the control system. TICS provides the basic CORBA objects for antenna control in horizontal and equatorial coordinates. In addition to that, there are utilities to record several kinds of time stamped monitor points into a database (MySQL) and to perform optical pointing runs.
The ACS and TICS packages fulfill the requirements of common network communication, automatic monitoring, real-time tracking and remote observing. The overarching software to use all hardware devices in a coordinated way necessary for astronomical observations was developed by the APEX software development group ([3]). This included defining the instrument and device interfaces ([4], [2]) and the raw data format interface (MBFITS, [5]).

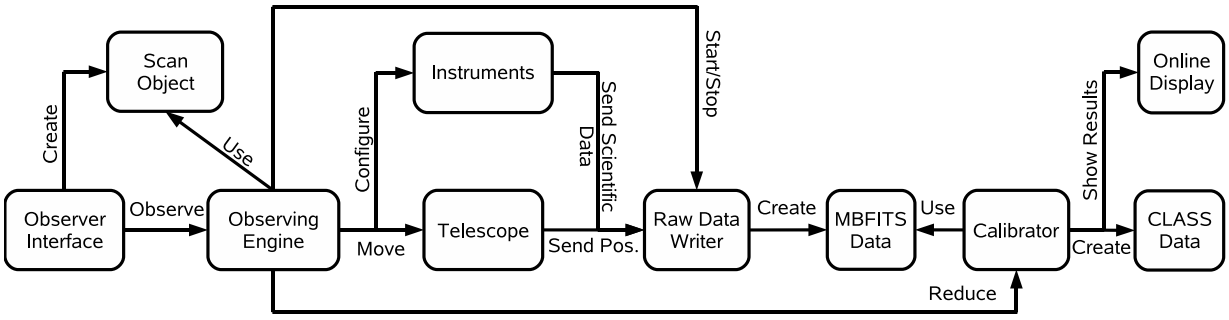


Figure 2.1: *This diagram illustrates the APECS observing pipeline structure. The astronomer submits a request for a scan – encoded as a so-called Scan Object – to the Observing Engine which then coordinates all hardware and software tasks to perform the observation. It sets up the instruments, moves the telescope to the desired position and starts the data recording. The Raw Data Writer collects the data streams and creates an MBFITS file. After each subscan the Calibrator provides calibrated data and shows results on the online display for user feedback.*

The interfaces for common instruments such as frontends or continuum and spectral backends are kept generic so that new instruments can be easily added to the system. The CORBA side of the interfaces is automatically generated from the interface files. Simulators can be used to run a full control system under Linux without the need for real hardware.
The observer level of APECS provides a scripting language for observing ("`apecs`") (a graphical user interface ("`xapecs`") is under development), the central Observing Engine to coordinate all devices and processes ("`apexObsEngine`"), the online MBFITS raw data writer ("`apexOnlineFitsWriter`") and the online data calibrator ("`apexOnlineCalibrator`") program to automatically perform the atmospheric corrections and provide CLASS (Gildas software) data with the $T_A^*$ temperature scale ([7]). All scans are automatically logged as XML and HTML files using the `apexObsLoggerServer`.
In addition, there are a number of monitoring tools to see the antenna positions ("`apexAntMount`"), the

Sun avoidance zone ("`plotAvoidance`"), the scan status ("`scanStatus`") and the values and possible alarm states of all available monitor points ("`apexStatusDisplay masterStatus.MasterStatus`").
These APECS core components are organised as a pipeline system (see fig. 2.1). Observations are defined using so-called Scan Objects which contain the full description of the next observation, i.e. the instrument setup details, target coordinate information and the desired observing patterns. The Scan Objects, that are created by the observer command line interface, are sent to the Observing Engine which sets up all necessary devices, controls the data acquisition and triggers the online data calibration, reduction and display.
APECS applies a relativistic Doppler correction to spectral line data based on the SLALIB library as of 2003. We checked the velocities against the Stumpff library used at the Effelsberg telescope. The two libraries agree to within 0.03 km/s. We chose SLALIB because of its slightly more accurate algorithms taking e.g. lunar effects into account.
Spectral line data is being calibrated to the $T_A^*$ temperature scale using Juan Pardo's ([6]) atmospheric library ATM. The calculations are made using Planck temperatures since the Rayleigh-Jeans approximation fails in the submm regime. Note, however, that the CLASS temperature scale is using the Rayleigh-Jeans approximation for compatibility reasons.

# Chapter 3

# Observing with the APEX Telescope

## 3.1 Introduction

Observations are performed using the APECS client computers, currently mainly `observer3`. The client observing sessions are started in the main VNC server (at `observer3:1`). The VNC servers are accessed through VNC viewers from the thin clients in the control rooms in Sequitor or at Chajnantor or externally via tunneling into the APEX network. When using your own computer be sure to use one of the recommended VNC viewer implementations (e.g. RealVNC v4.1.*) with the `-shared` option to avoid taking away other viewers' access.
Note that one **must not** use any of the APECS server computers (`control3`, `instruments3`, `display2` or `opt3`) for observations as their CPU resources are needed by the APECS services. One also **must not** run any other CPU intensive programs such as browsers, web cam displays, etc. on the servers. All APECS computers run under Linux (currently Scientific Linux 6.6 due to the use of ACS 2014.6).
The APECS startup procedures have been split into one server and two client scripts. The server script (`restartAPECSServers`) is started once on `control3` and brings up all server programs necessary for observing. This part is done by the APEX operators or APEX astronomers on duty using the `apex` operations account in the `control3:1` VNC server. After that server startup, clients can use the APECS services. It is possible to run several client sessions in parallel for different projects.
The operators will also create a new account for you based on your project ID. You must use that account for observing since all data products and log files are produced in separate project directories which are only accessible to the project account. Personal `bash` setups should be made in `$HOME/.acs/.bash_profile.$USER` to avoid overwriting `APECS` settings.
After logging on to one of the APECS client computers, you will find three symbolic links to your project's data:

- The `rawdata` link points to the directory containing the current `MBFITS` raw datasets on a computer at Chajnantor. The MBFITS dataset directories are named `APEX-<Scan number>-<YYYY-MM-DD>-<Project ID>` where the scan number is a sequential number starting with "1" on January 1st every year and the project ID is defined as described in section 3.2.1.1.
- The `scidata` link points to the directory containing the project's calibrated CLASS files at Chajnantor. The file naming scheme is `<Project ID>-<YYYY-MM-DD>.apex`, i.e. you will find one file per day. The spectra are identified using the CLASS `TELESCOPE` header variable. Due to the variable length restriction of 12 characters, we had to use acronyms to specify the frontend and backend names and the pixel and baseband numbers as defined in MBFITS ([4]). The spectra naming scheme is `AP-<2-letter frontend acronym><2-digit pixel number>-<2-letter backend acronym><2-digit baseband number>`. The frontend and backend acronyms are given in tables 3.4 and 3.5.
- The `obslogs` link points to the directory containing the project's automatically generated observation logs in XML and HTML format by the `apexObsLoggerServer`. The files are named `APECS-<Project ID>-<YYYY-MM-DD>.obslog` and `APECS-<Project ID>-<YYYY-MM-DD>.html`. The `apexObsLoggerClient` application will display today's project specific logs in a window and allow editing the user comments while observing. System comments are given in square brackets.

A file menu allows to view older logs of the same project. The log files are also transferred to the project database for remote access. The logs are part of the final data package that the PI will receive.

These links point to the local data directories depending on whether the machine is located at Chajnantor or in Sequitor. This avoids transferring large amounts of data over the network. The data and log files are first stored on a RAID at Chajnantor. They are transferred to Sequitor once per hour. For data inspection immediately after the observations one must therefore use the Chajnantor data reduction computer (`chajdr` (=`paruma`)). For full data reduction using data taken over longer periods one must use the Sequitor machine (`seqdr` (=`paniri`)).
Note that all of the above directories and files belong to the `apexdata` operations account. You cannot modify any of those files or create new ones within those directories as they are the base of the APEX archive. Make new directories and files off of your home directory and work from there. Please do *not* start applications inside the system directories since you do not have write permission there.
Apart from the data directories, there are a number of init files for the GILDAS software and the `lpoint.class` macro to reduce line pointings (type `@lpoint` in your CLASS session and enter the parameters asked for by the script; the line pointing fit will later be done automatically by the Online Calibrator).
If you log into a full KDE session, there is also a `Desktop` setup with the `APECS_XTerm"` icon. Note that the normal KDE terminals need to be started with the `-ls` option to correctly load the APECS paths.
To start your observing session type `restartAPECSMonClients` in one window. This will start several monitoring programs. Then type `restartAPECSObsClients` to start the necessary observing programs including the `apecs` CLI. The latter can also be simply started by typing `apecs` in a terminal window.
The observing clients include a default log message display showing the observer related logs only. The operator logs are shown in the `control3:1` VNC server. The calibrator messages are shown in the calibrator client GUI. If you want to see a different message selection, then you can start the "`apecsLog`" application with different filters:

| Logging GUI command | Filtered messages |
|---|---|
| apecsLog | Observer logs |
| apecsLog obs | Observer logs |
| apecsLog ops | Operator logs |
| apecsLog obops | Operator and Observer logs |
| apecsLog cal | Calibrator logs |
| apecsLog obopscal | All APECS logs |
| apecsLog all | All APECS logs |
| jlog | All APECS and technical logs |

Table 3.1: APECS logging GUI commands.

To observe with APEX one uses the "`apecs`" scripting language. "`apecs`" is based on "`(I)Python`" ([8]) and provides a number of special commands to set up several different standard observing modes. Normal Python programmatic structures and macro capabilities can be used from within "`apecs`". Note that `IPython` allows to write commands which begin directly at the prompt without the usual brackets "()" that Python requires. However, "apecs" macros that define user commands (see section 3.4) need to be written using the brackets.
The `apecs` commands lead to the creation of a so called "Scan Object" that is then sent to the "Observing Engine". The "Scan Object" contains all the setup information needed to control the instruments and the telescope. It is also being used for status displays and to fetch information needed for the MBFITS raw data files.
The current observing status will be shown in the "Scan Status" window. Apart from the scan progress it also shows antenna and environmental conditions and signals warnings as inverted yellow text and critical conditions as inverted red text. Immediate observer or operator attention is required for the latter ones.
To process your data, please use the GILDAS software for spectral line data and BoA or Crush for bolometer data. Please run any data reductions on the "paruma" or "paniri" computers.
At the end of the observing session please stop all clients using the `stopAPECSMonClients` and `stopAPECSObsClients` scripts before logging out.

## 3.2 apecs commands

The apecs commands are divided into several groups to handle source setup, instrument setup and pattern setup. In addition to that there are commands for various calibrations and some miscellaneous commands. Type "help(apecs)" at the "APECS>" prompt to get an overview and "help(command)" to get a description of the individual commands and their current default values. User parameters are usually taken as the next default except for a few cases where this does not make sense (e.g. "source", "pcorr", "fcorr", etc.).
APEX does have a 30 degree Sun avoidance zone. There is an automatic avoidance mechanism that moves the telescope around the Sun if necessary. This mechanism works only when using apecs commands. Observers **must not** move the telescope with the apexAntMount GUI since it does **not** know about the Sun avoidance zone !

### 3.2.1 General

#### 3.2.1.1 project_id

```
project_id( '<ID>')
```

Set the APEX project ID for use in the MBFITS raw data and CLASS scientific data files, and in the logs. It is very important to set this correctly at the beginning of each session in order to be able to identify your data later on ! apecs asks for the ID when is is started.
The project ID is assigned by the program committees or by the APEX station manager. The project ID "O-[K]PP.C-NNNN[R]-YYYY" is structured as follows:

- O is the origin of the project:
  - M: MPIfR
  - E: ESO
  - O: OSO
  - G: Germany (Verbundforschung)
  - C: Chile
  - P: PI Project
  - X: External (e.g. Berkeley)
  - K: Key program
  - A: APEX (staff time)
  - T: Technical / Maintenance / Calibration

K PP is the ESO proposal identifier (K=0 (regular), 1 (large), 2 (DDT), 3 (short), 4 (calibration) (optional; omit for non-ESO projects)) and period (PP) (may not apply to the other partners; use "00" in that case)

- C is the (ESO) category (A-F,L) (may not apply to the other partners; use "F" in that case):
  - A: COSMOLOGY
  - B: GALAXIES AND GALACTIC NUCLEI
  - C: INTERSTELLAR MEDIUM, STAR FORMATION and PLANETARY SYSTEMS
  - D: STELLAR EVOLUTION
  - E: UNDEFINED
  - F: UNDEFINED
  - L: CALIBRATION
- NNNN is a sequential number
- R is the ESO run ID letter (optional but regularly used for ESO, OSO and Chilean projects; omit for others)
- YYYY is the year

#### 3.2.1.2 operator_id

```
operator_id( '<operator initials>')
```

Set the operator ID for use in the MBFITS raw data file and in the logs.

#### 3.2.1.3 observer_id

```
observer_id( '<observer initials>')
```

Set the observer ID for use in the MBFITS raw data file and in the logs.

#### 3.2.1.4 show

```
show( )
show( <Scan object>)
```

Show the current observing mode setup. Optionally, one can pass a user defined scan object to show its setup.

#### 3.2.1.5 load

```
load( )
```

Configure the instruments without changing the telescope status. This is usually used to initialize devices for upcoming manual interactions such as tuning a non-remote-controlled receiver.

#### 3.2.1.6 go

```
go( )
```

Move the telescope to the desired center position and set up the instruments. Usually used to prepare the instrument setup for manual receiver tuning.

#### 3.2.1.7 track

```
track( )
```

Track the currently defined source without observing while still configuring the instruments. The telescope keeps tracking until the next scan command arrives.

#### 3.2.1.8 cancel

```
cancel( )
```

Cancel the ongoing scan.

#### 3.2.1.9 exec_apecs_script

```
exec_apecs_script( '<File name>')
```

Execute an "apecs" script from the official version controlled area.

#### 3.2.1.10 save_history

```
save_history( '<File name>')
```

Save the apecs command history into a file. The file may be edited and used as a macro later on. Macros are executed using the execfile command.

#### 3.2.1.11 save_defaults

```
save_defaults( '<File name>')
```

Not yet implemented. Will save the current user command defaults to a file.

#### 3.2.1.12 load_defaults

```
load_defaults( '<File name>')
```

Not yet implemented. Will load user command defaults from a file.

#### 3.2.1.13 reset_defaults

```
reset_defaults( )
```

Not yet implemented. Will load system defaults for all commands.

#### 3.2.1.14 observe

```
observe( )
observe( <Scan object>)
```

The internal method to submit a scan to the "Observing Engine". May be used to submit non-standard "Scan Objects".

#### 3.2.1.15 continuous_data

```
continuous_data( 'on' | 'off')
```

Switch between subscan based ('off') or continuous ('on') data taking during the whole pattern, i.e. also when turning around to the next row or column, etc. Turning this mode on is recommended only for bolometer array observations. For all other observing setups, the online and offline data calibration and reduction would no longer work.

#### 3.2.1.16 skip_hardware_setup

```
skip_hardware_setup( 'on' | 'off')
```

Select whether or not hardware should be set up for a scan. If the flag is set, then hardware is not configured if the setup is identical to the one used in the preceeding scan and if that scan has been completed less than 30 minutes ago. This saves scan overhead times, but it is somewhat risky as the correct setup is no longer guaranteed. One needs to take care that nobody modifies hardware parameters manually. The "go" command ignores this flag and always enforces a hardware setup.

#### 3.2.1.17 remote_control

```
remote_control( 'on' | 'off')
```

Switch apecs remote control mode on or off. This mode allows to send observing commands via a UDP socket connection on port 22122 to the current apecs session. This is mainly used for VLBI mode where the field system controls the telescope according to the schedule.
The apecs prompt indicates whether the remote control mode is active.

### 3.2.2 Catalogs

Define user source and line catalogs.

#### 3.2.2.1 sourcecats

```
sourcecats( '<File name>')
sourcecats( ['<File name 1>',
            '<File name 2>',
            ...])
```

Define the paths to user source catalog files in IRAM PdB format to specify source coordinates (equatorial J2000 or horizontal; no other systems are currently supported) and radial source velocity (LSR(K), radio convention). Table 3.2 shows an example of a source catalog. The entries must be stored in a simple text file in the given format.

| Source name(s) | System | Epoch | Lambda (RA or Az) | Beta (Dec or El) | Velocity frame | Velocity |
|---|---|---|---|---|---|---|
| Orion\|BN-KL | EQ | 2000.0 | 05:35:14.16 | -05:22:21.5 | LSR | 8.0 |
| Stow\|Park | HO | | 180:00:00.00 | 15:00:00.0 | | |

Table 3.2: Example of an APEX source catalog.

#### 3.2.2.2 ephemerides

```
ephemerides( '<File name>')
ephemerides( ['<File name 1>',
             '<File name 2>',
             ...])
```

Not yet implemented. Will define the paths to user ephemeris files in xephem "edb" format specifying orbital elements of solar system objects. Note that all known larger solar system objects (planets, moons, comets, asteroids) are automatically known by APECS even without such user ephemeris files.

#### 3.2.2.3 linecats

```
linecats( '<File name>')
linecats( ['<File name 1>',
          '<File name 2>',
          ...])
```

Define the paths to user line catalog files to specify transition details. Table 3.3 shows an example of a line catalog. The entries must be stored in a simple text file in the given format.

| Transition name | Frequency | Unit | Sideband |
|---|---|---|---|
| CO(4-3) | 461.040750 | GHz | LSB |
| CO(3-2) | 345.795969 | GHz | USB |

Table 3.3: Example of an APEX line catalog.

### 3.2.3 Instruments

This group of commands is used to define the instrument(s) to be used for the next scan. Instruments are composed of a frontend and a backend, thus they are also called "frontend-backend combinations" or FEBEs. There are bolometer (continuum) frontends and heterodyne frontends. Bolometers can be connected to continuum backends only. Heterodyne frontends can be connected to continuum backends (typically used for calibration, pointing, focus and skydip) and to spectral line backends (for single point integrations or maps).
Table 3.4 gives an overview of the frontends, table 3.5 shows the backends and their specifications. The CLASS acronyms in both tables are being used to create the CLASS header variable `TELESCOPE` according to the naming scheme `AP-<2-letter frontend acronym><2-digit pixel number>-<2-letter backend acronym><2-digit baseband number>`. Finally, tables 3.6, 3.7 and 3.8 show the possible FEBE combinations.

| Frontend | CLASS Acronym | Type | Receiver cabin | Number of pixels | Tuning Range [GHz] | Bandwidth [GHz] | Status |
|---|---|---|---|---|---|---|---|
| HET230 | H2 | Heterodyne (SSB) | A | 1 | 211–275 | 4 | Facility |
| HET345 | H3 | Heterodyne (SSB) | A | 1 | 275–370 | 4 | Facility |
| HET460 | H4 | Heterodyne (DSB) | A | 1 | 385–500 | 4 | Facility |
| HET1300 | H1 | Heterodyne (DSB) | A | 1 | 1250–1390 | 2 | Facility |
| FLASH345 | F3 | Heterodyne (2SB) | A | 2[1] | 272–377 | 2 × 4 | PI |
| FLASH460L | FL | Heterodyne (2SB) | A | 2[1] | 385–495 | 2 × 4 | PI |
| FLASH460H | FH | Heterodyne (2SB) | A | 2[1] | 460–510 | 2 × 4 | PI |
| CHAMP690 | C6 | Heterodyne (SSB) | B | 7 | 620–720 | 4 | PI |
| CHAMP810 | C8 | Heterodyne (SSB) | B | 7 | 790–950 | 4 | PI |
| PI460 | P4 | Heterodyne (DSB) | A | 1 | 450–490 | 2 | PI |
| PI1100 | P1 | Heterodyne (DSB) | A | 1 | 967–1042 | 4 | PI |
| ZSPEC | ZS | Heterodyne | A | 2[2] | 180–300 | 120 | PI |
| ZEUS2 | Z2 | Heterodyne | A | 10 | 350–1500 | 40 | PI |
| SUPERCAM | SC | Heterodyne (DSB) | C | 64 | 345.6–345.9 | 64 × 0.24 | PI |
| GARD180 | G1 | Heterodyne (2SB) | A | 4[3] | 159–211 | 4 × 4 | PI |
| LABOCA | LB | Bolometer | C | 295 | 345 | ∼60 | Facility |
| SABOCA | SB | Bolometer | C | 37 | 850 | 50 | Facility |
| BOLOSZ | SZ | Bolometer | C | 331 | 150 | ∼50 (TBC) | PI |
| ARTEMIS450 | A4 | Bolometer | C | 2304 | 666 | N/A | PI |
| ARTEMIS350 | A3 | Bolometer | C | 2304 | 866 | N/A | PI |
| ARTEMIS200 | A2 | Bolometer | C | 1152 | 1499 | N/A | PI |
| BOLOB | BB | Bolometer | B | 37 | 230 | 80 | PI |
| AMKID350 | M3 | Bolometer | C | 21600 | 850 | 60 | PI |
| AMKID870 | M8 | Bolometer | C | 3520 | 345 | 60 | PI |
| HOLO | HO | Holography | C | 1 | 92.4 | 0.56 | Techn. |

Table 3.4: APEX heterodyne and continuum frontends. The receiver cabins are named A (left Nasmyth), B (right Nasmyth) and C (Cassegrain).

[1] One pixel per sideband
[2] Virtual dual pixel setup to capture amplitude and phase
[3] One pixel per sideband and per polarization

| Backend | CLASS Acronym | Type | Bandwidths per input [MHz] | Number of inputs | Numbers of spectral channels | Status |
|---|---|---|---|---|---|---|
| AFFTS | AF | Spectral | 1500 | 32 | 8192, 4096, ... 4, 2, 1 | PI |
| XFFTS | XF | Spectral | 2500 | 8 | 65536, 32768, 1 | PI |
| XFFTS2 | X2 | Spectral | 2500 | 4 | 32768, 1 | Facility |
| ZSPECBE | ZB | Spectral | 120000 | 3 | 160, 62, 1 | PI |
| ZEUS2BE | ZE | Spectral | 160000 | 1 | 40, 24, 12, 1 | PI |
| PBE_A | PA | Continuum | FE/IF bandwidth | 8 | 1 | Facility |
| PBE_B | PB | Continuum | FE/IF bandwidth | 8 | 1 | Facility |
| PBE_C | PC | Continuum | FE/IF bandwidth | 8 | 1 | PI |
| PBE_D | PD | Continuum | FE/IF bandwidth | 8 | 1 | PI |
| PBE_E | PE | Continuum | FE/IF bandwidth | 8 | 1 | PI |
| PBE_F | PF | Continuum | FE/IF bandwidth | 8 | 1 | PI |
| ABBA | AB | Continuum | FE bandwidth | 320 | 1 | Facility |
| SZACBE | ZA | Continuum | FE bandwidth | 331 | 1 | PI |
| SZDCBE | ZD | Continuum | FE bandwidth | 331 | 1 | PI |
| BEAR1 | B1 | Continuum | FE bandwidth | 5632 | 1 | PI |
| BEAR2 | B2 | Continuum | FE bandwidth | 1664 | 1 | PI |
| AMKID350BE | M3 | Continuum | FE bandwidth | 25120 | 1 | PI |
| AMKID870BE | M8 | Continuum | FE bandwidth | 25120 | 1 | PI |

Table 3.5: APEX spectral and continuum backends

| | AFFTS (MPIfR) | XFFTS (MPIfR) | XFFTS2 | ZSPECBE (ESO) | ZEUS2BE (MPIfR) | SCBE (ESO) |
|---|---|---|---|---|---|---|
| HET230 | – | – | X | – | – | – |
| HET345 | – | – | X | – | – | – |
| HET460 | – | – | X | – | – | – |
| HET1300 | – | – | X | – | – | – |
| FLASH345 | – | X | – | – | – | – |
| FLASH460L | – | X | – | – | – | – |
| FLASH460H | – | X | – | – | – | – |
| CHAMP690 | X | – | – | – | – | – |
| CHAMP810 | X | – | – | – | – | – |
| PI460 | X | – | – | – | – | – |
| PI1100 | X | X | – | – | – | – |
| ZSPEC | – | – | – | X | – | – |
| ZEUS2 | – | – | – | – | X | – |
| SUPERCAM | – | – | X | – | – | X |
| GARD180 | – | X | – | – | – | – |
| HOLO | – | – | – | – | – | – |

Table 3.6: Possible FEBE combinations between the APEX heterodyne frontends and the spectral backends.

#### 3.2.3.1 `frontends`

```
frontends( <Frontend name>)
frontends( <Frontend name 1>,
           <Frontend name 2>,
           ...)
frontends( '<Frontend name>')
frontends( ['<Frontend name 1>',
           '<Frontend name 2>',
           ...])
```

Select the frontends to be used for the next scan. This command reloads all pointing models from files. Those models will become active in the following scan.

#### 3.2.3.2 `<Frontend name>.feeds`

```
<frontend name>.feeds( ref='default' | <Reference pixel number>,
                       select='all' | [<Pixel number list >] | 'circle<radius>' |
                              '<Specific geometry keyword>')
```

Configure the frontend feed setup. The reference feed number defined by "ref" shifts the receiver pointing model to center the observations on the new feed and tells the online display software which data to show. One needs to provide a number or the string "default" to go back to the original value.
The "select" parameter allows to restrict the number of feeds to be used. One can either specify an explicit list of feeds (e.g. [1,2,4,6]) or use specific geometric selections like "wedge1", "wedge2", etc. or "circle¡radius¿" (all pixels within the radius given in arcsec from the coordinate system center).
Currently, the following specific geometries are defined:

- BOLOSZ: 'wedge1', 'wedge2', ..., 'wedge6' (selects the corresponding wedge of pixels).

#### 3.2.3.3 `<Continuum frontend name>.configure`

```
<Continuum frontend name>.configure( gain=<Amplifier gain>)
<Continuum frontend name>.configure( polarimeter='s' | 'h' | 'v' | '')
```

| | PBE_A | PBE_B | PBE_C (MPIfR) | PBE_D (MPIfR) | PBE_F (MPIfR) |
|---|---|---|---|---|---|
| HET230 | X | – | – | – | – |
| HET345 | X | – | – | – | – |
| HET460 | X | – | – | – | – |
| HET1300 | X | – | – | – | – |
| FLASH345 | – | – | – | – | X |
| FLASH460L | – | – | – | – | X |
| FLASH460H | – | – | – | – | X |
| CHAMP690 | – | – | X | – | – |
| CHAMP810 | – | – | – | X | – |
| PI460 | X | – | – | – | – |
| PI1100 | X | – | – | – | X |
| ZSPEC | – | – | – | – | – |
| ZEUS2 | – | – | – | – | – |
| SUPERCAM | X | – | – | – | – |
| GARD180 | – | – | – | – | X |
| HOLO | – | X | – | – | – |

Table 3.7: Possible FEBE combinations between the APEX heterodyne frontends and the continuum backends.

| | ABBA | SZACBE (MPIfR) | SZDCBE (MPIfR) | BEAR1/2 (ESO) | PBE_E (MPIfR) | AMKID350BE (MPIfR) | AMKID870BE (MPIfR) |
|---|---|---|---|---|---|---|---|
| LABOCA | X | – | – | – | – | – | – |
| SABOCA | X | – | – | – | – | – | – |
| BOLOSZ | – | X | X | – | – | – | – |
| ARTEMIS450 | – | – | – | X | – | – | – |
| ARTEMIS350 | – | – | – | X | – | – | – |
| ARTEMIS200 | – | – | – | X | – | – | – |
| BOLOB | – | – | – | – | X | – | – |
| AMKID350 | – | – | – | – | – | X | – |
| AMKID870 | – | – | – | – | – | – | X |

Table 3.8: Possible FEBE combinations for the APEX continuum frontends.

Configure the continuum frontend. The gain parameter is used by some frontends. Typical values are 1, 2, 4, 8, etc. The parameter is only used if there is an amplifier CORBA object present in the system. The polarimeter configuration parameter is used to set the filter wheel orientation for instruments that support this type of observation. Valid values are 'S' (sky), 'H' (horizontal), 'V' (vertical) and '' (no action).

#### 3.2.3.4 <Heterodyne frontend name>.configure

```
<Heterodyne frontend name>.configure( mode='dsb' | 'ssb' | '2sb',
                                      ratios=[<Image to signal band gains for feed 1>,
                                      <... for feed 2>, ..., <... for feed N>],
                                      fthrow=<Frequency switching throw> |
                                      (<Phase 1 throw>, <Phase 2 throw>),
                                      harmonic=<Number>,
                                      doppler='on' | 'off',
                                      tuningmode='<mode>')
```

Configure basic heterodyne frontend parameters. The mode can be DSB, SSB or 2SB. The ratios are the linear image to signal band gain ratios for all feeds in numerical order. The frequency switching throw(s) are given in MHz. Usually, symmetric switching (±fthrow) is assumed. Asymmetric switching can be commanded using a list of throws (e.g. (-10.0,15.0)). The offsets are being used for the frequency switching observing mode which is activated by the `fsw` command. Note that the Observing Engine automatically chooses the recommended harmonic number if the user sets this to zero, which is the default. A non-zero number will override the system defaults.
The doppler parameter controls whether a Doppler correction is being applied for this frontend. It can be ON or OFF.
The tuning mode is an optional string to control the receiver tuning. Details are given in the online help in the `apecs` CLI.

#### 3.2.3.5 <Heterodyne frontend name>.line

```
<Heterodyne frontend name>.line( name='<Line name>',
                                 frequency=<Number>,
                                 sideband='lsb' | 'usb',
                                 unit='GHz',
                                 cats='all' | 'user' | 'sys')
```

Select the line to be used for the heterodyne frontend tuning. The system will try to find the line in the catalogs if only the name is given. Reading from the line catalogs performs a left-sided match with upper or lower case and wildcards for remainder of the string. The 'cats' parameter allows to restrict the catalogs to be searched. In particular, one can switch off searching the system catalogs by specifying 'user'.
For 2SB receivers the line name in the opposite sideband is set to "<Line name>_OSB" unless an explicit name is given in the corresponding <Spectral backend name>_group.configure command.

#### 3.2.3.6 <Heterodyne frontend name>.derotate

```
<Heterodyne frontend name>.derotate( mode='ca' | 'ho' | 'eq',
                                     angle='full' | 'half' | <User angle in degrees>)
```

Define derotation setup of array receivers. The angle in degrees defines the rotation of the array relative to the selected coordinate system. This is used to select the spatial sampling setup. Alternatively, one can specify 'full' or 'half' to automatically select those sampling setups. The mode can be 'CA' (fixed, no derotation), 'HO' (horizontal derotation) or 'EQ' (equatorial derotation).

#### 3.2.3.7 <Frontend name>.backends

```
<Frontend name>.backends( <Backend name>)
<Frontend name>.backends( <Backend name 1>,
                          <Backend name 2>,
                          ...)
<Frontend name>.backends( '<Backend name>')
<Frontend name>.backends( ['<Backend name 1>',
                          '<Backend name 2>',
                          ...])
<Frontend name>.backends( '<IF path>-<Backend name>')
<Frontend name>.backends( ['<IF path 1>-<Backend name 1>',
                          '<IF path 2>-<Backend name 2>',
                          ...])
```

Connect continuum and/or spectral line backends to the selected frontends. Note that `apecs` automatically assigns backend inputs. In case of continuum backends, the wiring is not computer configurable and the inputs are selected according to the hardware setup stored in `apecs`. In case of spectral line backends, some connections are computer configurable (e.g. via an IF processor). `apecs` administers those resources

automatically and assigns a number of spectral line backend inputs according to the number of receiver feeds.
In both cases, the selected inputs are named a "backend section group". The <Frontend name>.backends command will print a group number which is needed later on when configuring the properties of a particular "backend section group".
Usually, one uses simple backend names (e.g. 'pbe_a', 'xffts2', etc.). Optionally, one can specifiy the IF chain name (e.g. 'if4:c1-xffts2') to bypass the automatic section group assignment and to control the exact sequence in which backend section groups are assigned.

#### 3.2.3.8 <Continuum backend name>.configure

```
<Continuum backend name>.configure( dumptime=<Time in seconds>, gain=<Gain (1, ..., 100)>)
```

Configure the continuum backend dump time and gain. The default dump time is 0.0 s which means that the system determines the time automatically. The dump time is given in seconds and can range from 0.0 to 4.0 s. The gain can be an integer between 1 and 100. It controls the dynamic range by scaling the input signal by 1/gain.

#### 3.2.3.9 <Continuum backend name>_group.configure

```
<Continuum backend name>_group.configure( group=<Group number>,
                                          offset=<Offset in MHz>,
                                          sections=[<List of backend sections>])
```

Configure a continuum backend section group. The necessary group number is being displayed by `apecs` when connecting the backends to the frontends. It can also be queried using the `show()` command.
The section group radio frequency offset is given in MHz. The default sections making up this group can be overridden if needed. They must be given as a list.

#### 3.2.3.10 <Spectral backend name>.configure

```
<Spectral backend name>.configure( dumptime=<Time in seconds>)
```

Configure the spectral backend dump time. The default is 0.0 s which means that the system determines the time automatically. The dump time is given in seconds and can range from 0.0 to 4.0 s.

#### 3.2.3.11 <Spectral backend name>_group.configure

```
<Spectral backend name>_group.configure( group=<Group number>,
                                         bandwidth=<Bandwidth in MHz>,
                                         numchan=<Number of channels>,
                                         offset=<Offset in MHz>,
                                         linename='<Alternative line name>',
                                         sections=[<List of backend sections>])
```

Configure a spectral line backend section group. The necessary group number is being displayed by `apecs` when connecting the backends to the frontends. It can also be queried using the `show()` command.
Bandwidth and section group radio frequency offset are given in MHz. The numbers of channels can be chosen among the possible values for the particular backend. Optionally, a line name can be given if the section group is centered on a different line than the main one defined for the frontend. Leave the line name empty for composite spectra so that one can simply add the spectra in CLASS. Finally, the default sections making up this group can be overridden if needed. They must be given as a list.

### 3.2.4 Target

Commands in this section are used to define the target coordinates.

#### 3.2.4.1 source

```
source( '<Name>')
source( name='<Name>',
        x='<Longitude>',
        y='<Latitude>',
        unit='arcsec' | 'arcmin' | 'hdms' | 'hms' | 'dms' | 'deg' | 'rad',
        system='eq' | 'ho',
        epoch=2000.0,
        velocity=<Velocity in km/s> | 'keep',
        frame='LSR' | 'HELIO' (not yet available),
        cats='all' | 'user' | 'sys')
```

The source command is used to define a center position in either horizontal or equatorial (J2000 only at this point) coordinates. The parameters may be specified manually or can be read from a catalog (see `sourcecats`). Reading from the source catalogs performs a left-sided match with upper or lower case and wildcards for remainder of the string.
The angles can be given as numerical values with units 'deg', 'arcmin' or 'arcsec', as string 'HH:MM:SS.ssss' with unit 'HMS' or as string 'DD:MM:SS.ssss' with unit 'DMS'. The special unit description 'HDMS' is interpreted as 'HMS' for longitude and 'DMS' for latitude.
The velocity='keep' option keeps the value of the previous source. This is convenient when switching between target and pointing sources in order to avoid modifying the tuning.
The 'cats' parameter allows to restrict the catalogs to be searched. In particular, one can switch off searching the system catalogs by specifying 'user'.
The major solar system objects are automatically recognized by their names. Ephemeris files for a number of other solar system objects (comets, etc.) are installed in the system. User ephemeris files in `xephem`'s "`edb`" format can be introduced into the system (ask one of the AODs or TIOs to copy them to right place before the observations begin).
Note that the `TICS` software always uses descriptive coordinates, i.e. the new coordinate system centered at the source is using great circles in the sky. This is different compared to many other telescope control systems but it makes sense for the observing, in particular for wobbler observations where the wobbler motion occurs in that coordinate system too.

### 3.2.5 Calibration

#### 3.2.5.1 calibrate

```
calibrate( mode='cold' | 'hot',
           time=<Time per subscan in seconds> | [<Time for subscan 1>,
           <Time for subscan 2>, ... ], autolevel='on' | 'off')
```

Perform a calibration scan. Default is a sequence of SKY-HOT-COLD. If the mode is set to 'HOT', only SKY-HOT is being observed. Auto-levelling should usually be turned on. For some instruments levelling takes a very long time. This step can be skipped by specifying autolevel='off'.
The "time" parameter defines the number of seconds per subscan. It can be a scalar number to be used for all subscans or a list of numbers to specify different times for the individual subscans on the different loads. The sequence of loads is SKY, HOT and optionally COLD.
The sky position is being set up using the position defined in the "reference" command except for wobbler mode where the wobbler reference position is being used.
Note that calibration scans automatically turn off the wobbler while calibrations within maps use it if wobbler mode has been selected.

#### 3.2.5.2 skydip

```
skydip( azimuth='current' | <Angle in degrees>,
        am_stop=<Number>,
```

```
        am_start=<Number>,
        points=<Number>,
        time=<Time per point in seconds>)
```

Perform a skydip scan at the given azimuth (default is the current value) between the two air masses defined in the command. Do not specify an air mass of exactly 1.0 to avoid problems with the antenna control system.

#### 3.2.5.3 point

```
point( length=<Number>,
       unit='arcsec',
       time=<Time per subscan in seconds>,
       mode='otf' | 'ras',
       points=<Number>,
       direction='x' | 'y')
```

Perform a cross scan in OTF or raster mode to check the local pointing near the source.

#### 3.2.5.4 pcorr

```
pcorr( )
pcorr( <Azimuth offset in arcsec> | '*' | 'f',
       <Elevation offset in arcsec> | '*' | 'f')
```

Correct the receiver pointing model. Usually the numbers are fetched automatically (≡ 'f') from the online calibrator. For line pointings one currently needs to run the reduction script (`lpoint.class`) manually offline and enter the numbers manually as displayed by the script. Specifying '*' keeps the previous value.

#### 3.2.5.5 pcorr_reset

```
pcorr_reset( )
pcorr_reset( force='n' | 'y' )
```

Reset the user pointing offsets to zero. The force='y' parameter allows skipping the interactive user confirmation.

#### 3.2.5.6 focus

```
focus( amplitude=<Amplitude in mm or arcsec>,
       time=<Time per subscan>,
       points=<Number of points>,
       axis='z' | 'y' | 'x' | 'xtilt' | 'ytilt',
       mode='pos' | 'neg' | 'sym')
```

Perform a focus scan to check the local focus point at the current elevation. The focus of the selected axis is moved in the range of `-amplitude` to `+amplitude` with the given number steps in between. The mode can be "pos"(itive) for increasing offsets, "neg"(ative) for decreasing offsets or "sym"(metric) for both, starting with increasing offsets. A measurement is taken for each focus setting and the online calibrator tries to fit a parabola to the data.

#### 3.2.5.7 fcorr

```
fcorr( )
fcorr( delta=<Offset in mm or arcsec> | '*' | 'f',
       axis='z' | 'y' | 'x' | 'xtilt' | 'ytilt',
       febe=<FEBE to use for the fit>,
       group=<Section group number>)
```

Correct the local focus setting for a given axis. The number and the axis are fetched automatically (≡ 'f') from the online calibrator if no parameters are given to `fcorr`. Specifying '*' keeps the previous value.

#### 3.2.5.8 fcorr_reset

```
fcorr_reset( )
fcorr_reset( force='n' | 'y' )
```

Reset the user focus offsets for the current frontends to zero. The force='y' parameter allows skipping the interactive user confirmation.

#### 3.2.5.9 use_foc_temp_corr

```
use_foc_temp_corr( useFocusTemperatureCompensation='on' | 'off' )
```

Define whether to use the automatic focus temperature correction for the X, Y and Z axes. Accepts values 'on' and 'off'. The default is 'on'. When switching the correction off, the temperature correction deltas are added to the user deltas. When switching it back on, the user is asked to choose absolute (just temperature correction and zero user deltas) or relative mode (modify current user deltas such that the sums with the current auto-correction are equal to the current user values).

#### 3.2.5.10 set_tilts

```
set_tilts( AN=<AN pointing model term>, AW=<AW pointing model term>)
```

Set tilt values in arcsec to override the current pointing model (base plus frontend) values completely. "reset_tilts" reverts back to the models. This command may be needed if the telescope tilts change on a short time scale of a few hours. Please ask the APEX staff about the `apecs` script to determine the new numbers.

#### 3.2.5.11 reset_tilts

```
reset_tilts( )
```

Revert back to the tilts defined by the pointing models.

#### 3.2.5.12 use_linear_sensors

```
use_linear_sensors( useLinearSensors='on' | 'off')
```

Define whether to use the linear sensor pointing correction. Accepts values 'on' and 'off'.

#### 3.2.5.13 use_tiltmeters

```
use_tiltmeters( useTiltmeters='on' | 'off', temp_comp='on' | 'off')
```

Define whether to use the tiltmeters pointing correction. Accepts values 'on' and 'off'. Optionally enable temperature compensation ('on' or 'off').

#### 3.2.5.14 set_cold_params

```
set_cold_params( frontend='<Frontend name>',
                 yfactors=[y-factor1, y-factor2, ...],
                 tcold=<Cold load temperature>)
```

Manually set the cold load parameters of a frontend in the online calibrator. To be used if the cold load malfunctions and the Y-factors are measured manually. The Y-factors must be given for all feeds. The cold load temperature must be specified in Kelvin.

#### 3.2.5.15 reset_cold_params

```
reset_cold_params( frontend='<Frontend name>')
```

Reset the cold load parameters of a frontend in the online calibrator.

### 3.2.6 Observing Patterns

Observing patterns are set up using the `TICS` strokes. There are linear, spiral and Lissajous strokes. By default all of the APECS patterns are composed of linear strokes, i.e. either pointed observations or linear OTF strokes. The `spiral` and `lissajous` commands can be used to set up spiral or Lissajous patterns instead of pointed observations (e.g. `on`, `raster`, `focus`, etc.). The stroke commands `linear`, `spiral` and `lissajous` are mutually exclusive.
Note that `TICS` sets up a new descriptive coordinate system centered on the source coordinates and that offsets are always applied along the great circles. This means that all patterns will look slightly different when being projected to encoder coordinates (azimuth / elevation) or equatorial coordinates (right ascension / declination).
The position angle, that can be specified in some observing pattern commands, follows the mathematical convention, i.e. positive values are assigned to the north to west rotation, thus $+45^{o}$ means NW in the astronomical coordinate system (see 3.1).

#### 3.2.6.1 offset

```
offset( x=<numerical x-Offset> | 'HH:MM:SS.ssss' | 'DD:MM:SS.ssss',
        y=<numerical y-Offset> | 'HH:MM:SS.ssss' | 'DD:MM:SS.ssss',
        unit='arcsec' | 'arcmin' | 'hdms' | 'hms' | 'dms' | 'deg' | 'rad',
        system='eq' | 'ho',
        epoch=2000.0)
```

Define the offset position for the next pattern to be centered on. The special 'hdms' unit is interpreted as 'hms' for the longitude offset and 'dms' for the latitude offset.
Note that the offset is interpreted in the rotated coordinate system if a non-zero angle is specified in mapping commands.
Note that observing a horizontal pattern on an equatorial offset position leads to shifting the user coordinate system to that offset position before applying the horizontal offsets.

#### 3.2.6.2 reference

```
reference( x=<numerical x-Offset> | 'HH:MM:SS.ssss' | 'DD:MM:SS.ssss',
           y=<numerical y-Offset> | 'HH:MM:SS.ssss' | 'DD:MM:SS.ssss',
           time=<Time in seconds>,
           on2off=<Number of On's per Off>,
           unit='arcsec' | 'arcmin' | 'hdms' | 'hms' | 'dms' | 'deg' | 'rad',
           mode='rel' | 'abs',
           system='eq' | 'ho',
           epoch=2000.0)
```

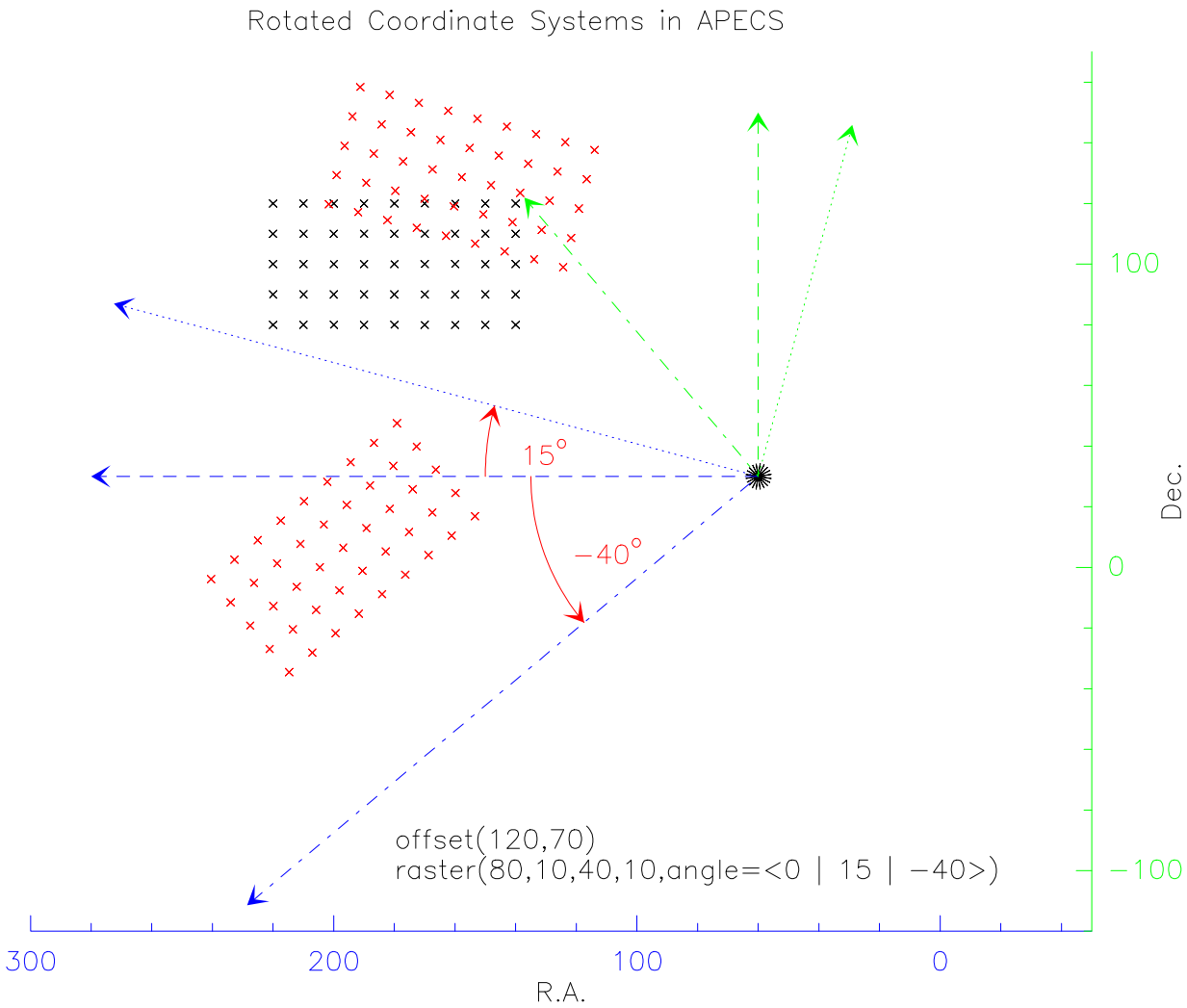


Figure 3.1: *Rotated coordinate systems in APECS shown for a raster command. The same angle definition is used for all other patterns.*

Define a sky reference position for any On-Off type observation. Units can be 'arcsec', 'arcmin', 'hdms', 'hms', 'dms', 'deg' or 'rad'. The special 'hdms' unit is interpreted as 'hms' for the longitude offset and 'dms' for the latitude offset. The mode can be 'rel'(ative) or 'abs'(olute). The reference positions are always taken before the target measurements.
If the time is set to 0.0, the system will automatically calculate the ideal off time based on the number of on's per off and the beam size: $t_{off} = \sqrt{N_{ON}} * t_{beam}$. For OTF maps the number of ON beams is calculated using the largest beam size of a given setup and the time per beam is the sum of the dump times across one beam.
The reference position is also used by the "calibrate" command to set up the sky subscan. Note that the reference position is not used for wobbler observations.

#### 3.2.6.3 use_ref

```
use_ref( 'on' | 'off')
```

Select whether to use the reference position in an observing mode. Note that the system automatically skips the reference if only continuum backends are connected (except for the "on" command) or if the wobbler is being used. Thus it is usually not necessary to change the default of 'on' for `use_ref`.

#### 3.2.6.4 on

```
on( time=<Time per point>,
    drift='no' | 'yes',
    feeds=[<feed number>, <feed number>, ...],
    offsets=[(<x1>, <y1>), (<x2>, <y2>), ..., (<xN>, <yN>)],
    offsets_unit='arcsec' | 'arcmin' | 'deg',
    on2cal=<Number of "on" subscans between calibrations>)
```

Perform a single point centered on the offset defined by the `offset` command. The time is given in seconds. Uses the reference if `use_ref` is "ON". Enabling the "drift" option leads to observing a fixed horizontal position equivalent to the mid-subscan astronomical source position. This is being re-calculated per subscan.
Specifying a list of feed numbers causes the setup of a sequence of "on" commands on the corresponding feed offset positions. This is usually used for wobbled dual-beam scans where the "off"-phase happens to fall onto another feed of the array receiver in use. Reset to [] to disable this option.
Specifying a list of offsets causes the setup of a sequence of observations pointed to those offsets relative to the position defined by the "source" and "offset" commands. The offset positions are given as a list of tuples, e.g. "[(10,20),(-20,40)]". The values are interpreted as angles given by the "offsets_unit" parameter. Reference positions are observed according to the settings in the "reference" and "use_ref" commands. Reset to [] to disable the "offsets" option.
Note that the "feeds" and "offsets" parameters are mutually exclusive.
For long scans calibrations can be inserted within the map for every "on2cal" "on" subscans. The setup makes sure that "ref"/"on" cycles are completed before the next calibration. Thus "on2cal" is the minimum number of subscans between calibrations. The calibrations are performed according to the settings in the "calibrate" command. Setting "on2cal" to 0 disables calibrations within maps. Note that an initial "calibrate" command is needed after tuning a receiver to auto-level the IFs.
Note that the resulting `CLASS` data are projected into encoder coordinates (horizontal system) or J2000 coordinates (equatorial system). Due to the use of descriptive coordinate systems in APECS, the projected values are not the same as the commanded descriptive values.

#### 3.2.6.5 raster

```
raster( xlen=<x-Length>,
        xstep=<x-Step>,
        ylen=<y-Length>,
        ystep=<y-Step>,
        time=<Time per raster point>,
        direction='x' | 'y',
        zigzag=1 | 0,
        angle=<Position Angle>,
        size_unit='arcsec' | 'arcmin' | 'deg',
        angle_unit='deg' | 'arcmin' | 'arcsec',
        system='eq' | 'ho',
        epoch=2000.0,
        mode='ordered' | 'jiggle',
        on2cal=<Number of "on" subscans between calibrations>)
```

Perform a rectangular raster pattern given by lengths and steps specified in 'size_unit' in both directions. The time is used per raster point. If zigzag==1, the raster is done bi-directionally, reducing the telescope overhead. The direction defines the fastest varying axis for an ordered setup. The angle defines the position angle of the map relative to the coordinate system. The sign of this angle follows the mathematical convention for all coordinate systems.
The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the raster points in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.

For long scans calibrations can be inserted within the map for every "on2cal" "on" subscans. The setup makes sure that "ref"/"on" cycles are completed before the next calibration. Thus "on2cal" is the minimum number of subscans between calibrations. The calibrations are performed according to the settings in the "calibrate" command. Note that an initial "calibrate" command is needed after tuning a receiver to auto-level the IFs.
Note that the resulting `CLASS` data are projected into encoder coordinates (horizontal system) or J2000 coordinates (equatorial system). Due to the use of descriptive coordinate systems in APECS, the projections are no longer simple rectangles.

#### 3.2.6.6 `hexa`

```
hexa( time=<Time per raster point>,
      sampling='full' | 'half' | 'extend',
      beamsize=0.0 | <Beam size>,
      unit='arcsec' | 'arcmin',
      feedsep=<Feed separation in number of beam sizes>,
      system='eq' | 'ho',
      epoch=2000.0,
      mode='ordered' | 'jiggle',
      on2cal=<Number of "on" subscans between calibrations>)
```

Perform a hexagonal raster pattern to map out the footprint area of a (hexagonal) array receiver (e.g. CHAMP+). The time is used per raster point.
The sampling can be "full", "half" or "extend" for fully / half sampled maps or to extend a half sampled map with the remaining points to obtain a fully sampled one. The beam size is automatically calculated from the highest sky frequency for the current scan if a value of 0.0 is given. Otherwise the non-zero user value is used.
The feed separation is the radial distance of neighboring feeds in number of beam sizes (for CHAMP+ this number is 2). The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the raster points in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.
For long scans calibrations can be inserted within the map for every "on2cal" "on" subscans. The setup makes sure that "ref"/"on" cycles are completed before the next calibration. Thus "on2cal" is the minimum number of subscans between calibrations. The calibrations are performed according to the settings in the "calibrate" command. Note that an initial "calibrate" command is needed after tuning a receiver to auto-level the IFs.
Note that the array derotation angle should be 0.0 in "eq" mode.

#### 3.2.6.7 `hexa25`

```
hexa25( time=<Time per raster point>,
        step=<Radial distance to neighbor>,
        unit='arcsec' | 'arcmin',
        system='eq' | 'ho',
        epoch=2000.0,
        mode='ordered' | 'jiggle',
        on2cal=<Number of "on" subscans between calibrations>)
```

Perform a hexagonal raster pattern to map out the footprint area of CHAMP+. The time is used per raster point.
The step size is the radial distance of neighboring points. The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the raster points in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.
For long scans calibrations can be inserted within the map for every "on2cal" "on" subscans. The setup makes sure that "ref"/"on" cycles are completed before the next calibration. Thus "on2cal" is the minimum number of subscans between calibrations. The calibrations are performed according to the settings in the

"calibrate" command. Setting "on2cal" to 0 disables calibrations within maps. Note that an initial "calibrate" command is needed after tuning a receiver to auto-level the IFs.
Note that the array derotation angle should be 0.0 in "eq" mode.

#### 3.2.6.8 otf

```
otf( xlen=<x-Length>,
     xstep=<x-Step>,
     ylen=<y-Length>,
     ystep=<y-Step>,
     time=<Time per OTF map point>,
     direction='x' | 'y',
     zigzag=1 | 0,
     angle=<Position Angle>,
     size_unit='arcsec' | 'arcmin' | 'deg',
     angle_unit='deg' | 'arcmin' | 'arcsec',
     system='eq' | 'ho',
     epoch=2000.0,
     mode='ordered' | 'jiggle',
     on2cal=<Number of "on" subscans between calibrations>)
```

Perform a rectangular "On-The-Fly" pattern given by lengths and steps specified in 'size_unit' in both directions. The telescope scans along map rows or columns with a speed defined by the step and time parameters of the selected direction while data is being written once per defined time interval. The time is used per OTF map point.
If zigzag==1, the OTF map is done bi-directionally, reducing the telescope overhead. The direction defines the scanning axis. The angle defines the position angle of the map relative to the coordinate system. The sign of this angle follows the mathematical convention for all coordinate systems. The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the OTF lines in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.
For long scans calibrations can be inserted within the map for every "on2cal" "on" subscans. The setup makes sure that "ref"/"on" cycles are completed before the next calibration. Thus "on2cal" is the minimum number of subscans between calibrations. The calibrations are performed according to the settings in the "calibrate" command. Note that an initial "calibrate" command is needed after tuning a receiver to auto-level the IFs.
Note that the steps along the scanning axis should be ideally chosen to be 0.45 times the current beam size. The beam will become elongated if the steps are chosen to be too large.
Note that the resulting CLASS data are projected into encoder coordinates (horizontal system) or J2000 coordinates (equatorial system). Due to the use of descriptive coordinate systems in APECS, the projections are no longer simple rectangles.

#### 3.2.6.9 drift

```
drift( azsize=<map size in azimuth>,
       elsize=<map size in elevation>,
       elstep=<step size in elevation>,
       lineartime=<time for linear stroke>,
       turntime=<time for turn-around>,
       mode='OPEN' | 'CLOSED',
       azunit='deg' | 'arcmin' | 'arcsec',
       elunit='arcmin' | 'deg' | 'arcsec')
```

Observe a rectangular azimuthal On-The-Fly drift pattern along lines of constant elevation with half circle turn-arounds to minimize microphonics through telescope accelerations. The pattern is centered on a horizontal source or around the average azimuth and elevation of an equatorial source. The source thus

drifts through the line of sight during the scan. Note that the source name is changed to <Original name>-DR_<Az>_<El> to show that a fixed position in horizontal coordinates was observed.
The extent of the pattern is given by the size of the linear portion of the azimuth stroke and a size in elevation. In "open" mode, rows spaced by the elevation step size are set up with zig-zag scanning. In "closed" mode, a "race-track" like closed loop pattern is set up. In this mode the elevation size can be zero to obtain a simple sweep pattern at one elevation value. Repetitions of these patterns are controlled with the "repeat" command. In case of the "race-track" pattern, all cycles are performed in one subscan. The durations (in seconds) of the linear stroke and the turn-around can be set individually, however the turn-around time must be at least 2 seconds to avoid extreme telescope accelerations.
The default backend dump time for drift scans is 0.5 s. This can be overridden by setting individual backend dump times using the backend setup commands.
This mode is mainly used for the SZ project to minimize signal modulations by the earth's atmosphere.

#### 3.2.6.10 repeat

```
repeat( <Number of pattern repeats>)
```

Set the number repeats for any given observing mode. Used to repeat a given pattern (e.g. a raster) <N> times. Note that `repeat` also applies to the pointing and focus commands. Selecting a new source or a new backend setup will reset the repeat count to 1.

### 3.2.7 Stroke Mode

#### 3.2.7.1 linear

```
linear( )
```

Select the linear stroke mode. This is the normal mode for all patterns. Alternatively, one can select spiral or Lissajous figures.
The linear mode is mutually exclusive with the spiral and Lissajous modes.

#### 3.2.7.2 spiral

```
spiral( r0=<Starting radius>, r_dot=<Radial veolcity>,
        phi_dot=<Angular velocity>,
        r0_unit='arcsec' | 'arcmin' | 'deg',
        r_dot_unit='arcsec/s' | 'arcmin/s',
        phi_dot_unit='deg/s' | 'arcmin/s')
```

Set up a spiral On-The-Fly pattern for use with other observing commands. The spiral is performed relative to the current offset position. It is defined by a start radius, a radial velocity and an angular velocity. The default backend dumptime 0.5 s. This can be overriden with the backend setup commands. The total time for the spiral is defined by the individual observing commands like "on", "raster", "point", etc.
The spiral mode is mutually exclusive with the linear and Lissajous modes.

#### 3.2.7.3 lissajous

```
lissajous( xlen=<x-Length>, ylen=<y-Length>,
           omega_x=<Angular velocity in x direction>,
           omega_y=<Angular velocity in y direction>,
           xlen_unit='arcsec' | 'arcmin' | 'deg',
           ylen_unit='arcsec' | 'arcmin' | 'deg',
           omega_x_unit='rad/s' | 'deg/s'
           omega_y_unit='rad/s' | 'deg/s')
```

Set up a Lissajous On-The-Fly stroke for use with other observing commands. The Lissajous pattern is performed relative to the current offset position. It is defined by extents and angular velocities along the x and y axes of the observing coordinate system.
The angular velocities are currently limited to 0.0 ... 0.11 rad/s.
The Lissajous mode is mutually exclusive with the linear and spiral modes.

### 3.2.8 Switch Mode

#### 3.2.8.1 tp

```
tp( )
```

Select total power mode. All subsequenct scans are performed without wobbler or frequency switching.

#### 3.2.8.2 wob

```
wob( amplitude=<Wobbler amplitude in arcsec>, rate=<Wobbler rate in Hz>,
     mode='sym' | 'neg' | 'pos' | 'lrrl', blank=-1 | <Blank time in ms>)
```

Select wobbler mode. All subsequent scans are performed using the wobbler with the parameters given here. The wobbler amplitude is given in arcsec, the wobbling rate in Hz. The mode can be "sym"(metric), "lrrl"(left-right-right-left symmetric), "neg"(ative) or "pos"(itive) to select the reference point location. The optional blank(ing) parameter allows to override the system setting of the blank time to be used for wobbler observations. The time is given in milli-seconds. A value of -1 selects the automatic mode.

#### 3.2.8.3 fsw

```
fsw( rate=<Frequency switching rate in Hz>, blank=-1 | <Blank time in ms>)
```

Select frequency switching mode. All subsequent scans are performed using frequency switching. The frequency switching rate is given in Hz. The frequency throws are defined individually for each frontend using the <heterodyne frontend name>.configure commands. The optional blank(ing) parameter allows to override the system setting of the blank time to be used for frequency switching observations. The time is given in milli-seconds. A value of -1 selects the automatic mode.

### 3.2.9 Antenna

#### 3.2.9.1 tolerance

```
tolerance( <Tolerance radius in arcsec>)
```

Define the required initial tracking accuracy. A subscan will only begin if the telescope is within the given radius for 2 or more consecutive time ticks of 48ms.

#### 3.2.9.2 park

```
park( )
```

Move the telescope to the stow position (South at 15 degrees elevation) and switch to SHUTDOWN mode without inserting the stow pins. This command should be used at the end of an observing session to park the telescope in a safe position.

#### 3.2.9.3 zenith

```
zenith( azimuth='current' | <Angle in degrees>)
```

Move the telescope to zenith. By default the scan is performed at the current azimuth. Optionally, an azimuth value can be given in degrees.

#### 3.2.9.4 stow

```
stow( )
```

Move the telescope to the stow position (South at 15 degrees elevation), insert the stow pins, switch to SHUTDOWN mode and stow the wobbler. Note that the Observing Engine will refuse to observe any further scan until the telescope is un-stowed. This command should be used in case of critical environmental conditions that could do harm to the telescope if it is not stowed.
Caution: You must use `unstow` to retract both the antenna and wobbler stow pins. Local operation at the ACU is not sufficient !

#### 3.2.9.5 unstow

```
unstow( )
```

Retract the antenna and wobbler stow pins and park the telescope at the stow position (South at 15 degrees elevation). The telescope is now prepared for observations. Note that the wobbler is *not* unstowed if the antenna pins are taken out at the ACU or via the `acu_wiu` script.

#### 3.2.9.6 stow_wobbler

```
stow_wobbler( )
```

Perform a direct stow of the wobbler. To be used only when recovering from any wobbler error conditions (see wobbler manual).

#### 3.2.9.7 init_wobbler

```
init_wobbler( )
```

Perform a direct unstow and initialization of the wobbler. To be used only when recovering from any wobbler error conditions (see wobbler manual).

#### 3.2.9.8 reset_wobbler

```
reset_wobbler( )
```

Perform a direct reset of the wobbler errors. To be used only when recovering from any wobbler error conditions (see wobbler manual).

#### 3.2.9.9 switch_c_optics

```
switch_c_optics( cabin='A' | 'B' | 'C')
```

Switch the Cassegrain optics to point to the given cabin ('A', 'B' or 'C'). Note that this can only be done when the telescope is pointing to zenith.

## 3.3 A typical Observing Session

The following sections show some example sequences of `apecs` commands to set up source and instruments and to perform some typical continuum and spectral line observations.

### 3.3.1 Source Setup

```
# Define a source catalog
sourcecats('user.cat')
# Define a line catalog
linecats('user.lin')
# Load a source from the catalog
source('bn-kl')
```

### 3.3.2 Continuum Instrument Setup

```
# Define frontend(s)
frontends('laboca')
laboca.configure(gain=2)
# Continuum backends
laboca.backends('abba')
# Select continuum backend gain
abba.configure(gain=4)
```

### 3.3.3 Spectral Line Instrument Setup

```
# Define frontend(s)
frontends(['pi460'])
# Define lines
pi460.line('CO(4-3)')
# Spectral backend setup.  Note the group numbers that the CLI displays.
# They can also always be queried using the show() command.
pi460.backends('affts')
# Configure the spectral backends
affts_group.configure(1, numchan=8192)
```

### 3.3.4 Initial Calibrations

```
# Select a submm calibrator
source('mars')
# Select continuum backend(s) also for heterodyne receivers.
flash345.backends('pbe_a')
flash460.backends('pbe_a')
# Perform a SKY-HOT-COLD measurement
calibrate()
# Perform an OTF cross scan.  The online calibrator automatically fits a
# Gaussian and provides the offsets to correct the pointing model.
point()
# Fetch the pointing offsets and apply them to the receiver pointing model.
pcorr()
# Perform a focus measurement (default "Z", i.e. radial focus)
focus()
# Fetch the focus offsets and apply them to the receiver focus position
# and auto-correct its pointing model for lateral focus shifts.
fcorr()
```

### 3.3.5 Continuum Observations

```
# Select frontends and continuum backends as shown above
# Select a science target
source('orion')
# Define a reference position for the calibration.
reference(-3600, 0)
# Perform a SKY-HOT-COLD measurement
calibrate()
# Perform an OTF map.  Note that the reference is not being used here
# since no spectral backend is connected.
otf(3600, 1, 200, 8, time=0.005, system='eq')
```

### 3.3.6 Spectral Line Observations

```
# Select frontends and spectral line backends as shown above
# Select a science target
source('orion')
# Define a reference position.  Note that the time will be automatically
# computed for the different observing modes if it is set to 0.0 in the
# reference command (this is the default).
reference(-3600, 0)
# Perform a SKY-HOT-COLD measurement
calibrate()
# Perform an OTF map.  Note that the reference is being used automatically
# if spectral backends are connected.
otf(400, 3, 30, 3, time=0.1, system='eq')
```

## 3.4 Macros and Loops

Macros are supported via `execfile`. The scripts must use standard Python syntax with brackets. To run a set of `apecs` commands, edit them into a text file and execute them from the `APECS>` prompt by typing `execfile '<filename>'`.
There are some default observing macros in the `$APECSROOT/share/apecs/` directory. To run them, one can use `exec_apecs_script('<Script name>')` or `execfile(APECS_SCRIPTS+'<Script name>')`. Currently, there are macros to set up SHFI ("`shfi_commands.apecs`"), FLASH+ ("`flash_commands.apecs`"), and CHAMP+ ("`champ_commands.apecs`") which provide high-level setup commands ("`setup_shfi`", "`setup_flash`", and "`setup_champ`") that take into account certain hardware limitations. It is recommended to use those macros when observing with the above frontends. Some of the macros also contain special observing commands. Check the online help in `apecs` for more details.
Often one wants to repeat a certain set of commands or macros in a loop. Use the Python programmatic structures to accomplish this. Note that Python is strict about keeping the same amount of indentation for a given loop or branch level.
One can also use Python variables to construct more complicated complex patterns alternating between calibrations and target observations. One example of using loops and variables would be the following set of OTF maps:

```
reference(-1800, 0, on2off=1, mode='rel', system='EQ')
# Loop between -60 and 120 in steps of 60.
# Note that xrange works with integer numbers.  Floating point numbers are
# provided by the Numeric.arange() function.
for yoff in xrange(-60, 121, 60):
    offset(20, yoff)
    calibrate()
    otf(270, 5, 45, 5, time=1, direction='x')
```

## 3.5 User defined Commands

More complex user commands can be defined in `apecs` via Python functions. For example, the switch to continuum backends and a subsequent pointing can be grouped together to define a new user command. The following example defines a special pointing command for FLASH observations. The two parameters `length` and `time` are defaulted to some values but can be overwritten like for normal `apecs` commands. Please consult the Python documentation at `www.python.org` for more information on programming. The command definition must be read into `apecs` using `execfile`.

```
def flash_point(length=180,time=30):
    frontends(['flash345','flash460'])
    flash345.backends('pbe_a')
    flash460.backends('pbe_a')
    point(length=length,time=time,mode='otf')
```

The newly defined commands can be used like any other `apecs` command.

## 3.6 Notes and Caveats

- Note that scans are submitted to a queue which is 2-deep, i.e. while the current scan is being executed, the next one will be kept on hold within the `apecs` CLI giving the user a chance to cancel the submission by typing <CTRL-C> (be careful though about <CTRL-C> at the normal prompt (see below)).
- Under ACS 2014.6 it is usually also safe to type <CTRL-C> at the normal `apecs` prompt. This does not lead to an immediate segmentation fault like in the old system under ACS 2.0.1. However, repeated typing of <CTRL-C> under certain circumstances can still crash the `apecs` CLI.
- Do not type `help(<command>())` (note the second pair of brackets) as this will execute the command rather than print help on it. In particular, one may inadvertently submit a new scan to the Observing Engine. `help` must be used with just the command name, i.e. `help(<command>)`.
- Do `not` use the IPython function "`run`" to run a script. It will **not** be executed in the correct `apecs` name space and will thus not work.
- Note that Python is strict about keeping the same amount of indentation for a given loop or branch level.
- Note that all APECS computers are running on TAI rather than UTC even though the time commands claim to display GMT (Linux does not have a TAI timezone). The local time is thus currently 34 seconds ahead of UTC. This number will increase by one whenever the next leap second will be introduced.
- For operators and astronomers on duty: Be **extremely** careful when using `apexAntMount expert` to move the telescope because it does not obey the Sun avoidance zone !

# Chapter 4

# Operating the APEX Telescope

## 4.1 Introduction

This section is aimed at APEX operators and (experienced) test observers. It describes how to start APECS and how to troubleshoot the system. The system is still evolving as development, debugging and improvements are still going on. Thus the content of this chapter will change with time and this document will be updated accordingly.
APECS is running on a number of control computers inside the antenna ("`abm`", "`opt3`"), in the control containers at Chajnantor (servers: "`control3`", "`instruments3`", "`display2`"; clients: "`observer3`", "`chajdr/paruma`") and in the base at Sequitor ("`seqdr/paniri`"). All of those computers, except `abm`, are running under Linux (currently Scientific Linux 6.6 due to using ACS 2014.6; `opt3` still uses Scientific Linux 5.8 with APECS 2.7 / ACS 12.1 due to the Linux device driver for the optical frame grabber card being unsupported for more recent Linux versions). The "`abm`" computer is a VME system running VxWorks 6.7.
In addition to the APECS computers there are a number of embedded system computers ("`apexcool`", "`cs-a1`", "`hfe-control`", "`affts`", "`xffts`", "`laboca`", "`abba`", "`if_control`", etc. and, to some extent, `instruments3`) to control the instruments. Those computers use different versions of Linux or Windows. Embedded system computers are under the responsibility of the respective system developers. The interface to the embedded system is realized via ASCII SCPI commands sent over UDP sockets ([1]).
Fig. 4.1 shows the deployment of the APECS software on the various computers inside the telescope and in the control rooms at Chajnantor and in Sequitor.

## 4.2 Starting the APECS servers

The APECS servers provide the necessary infrastructure to perform observations. They are being started via a single script called `restartAPECSServers`. This script needs to run on the central APECS machine called `control3`. It starts the core CORBA services and the APECS service applications such as Observing Engine, MBFITS Writer and Online Calibrator. Usually, a VNC session (`control3:1`) is used to start APECS so that one can check the server consoles remotely. One needs to connect to the VNC server via typing `vncviewer -shared control3:1`.
Before starting APECS, one should make sure that the machines are all idle and that the disks are not full. One should not run any other CPU intensive applications on the APECS server computers (`control3`, `display2`, `instruments3` and `opt3`). Make sure that there is no web cam display running on any of the core computers at the site at Chajnantor as those may use all the available CPU power.
For the server startup one needs to log on to `control3` as user `apex`. At the prompt on `control3` type `restartAPECSServers [-f]` to start APECS.
There is also a simulation mode in which the telescope is being simulated and the real telescope is not moved. Everything else is identical to the real setup. This mode can be started by typing `restartAPECSServers -s [-f]`. Note that the tracking in simulation mode is not as good as in the real hardware due to the 10 ms Linux scheduler not being commensurate with the 48 ms ticks used in

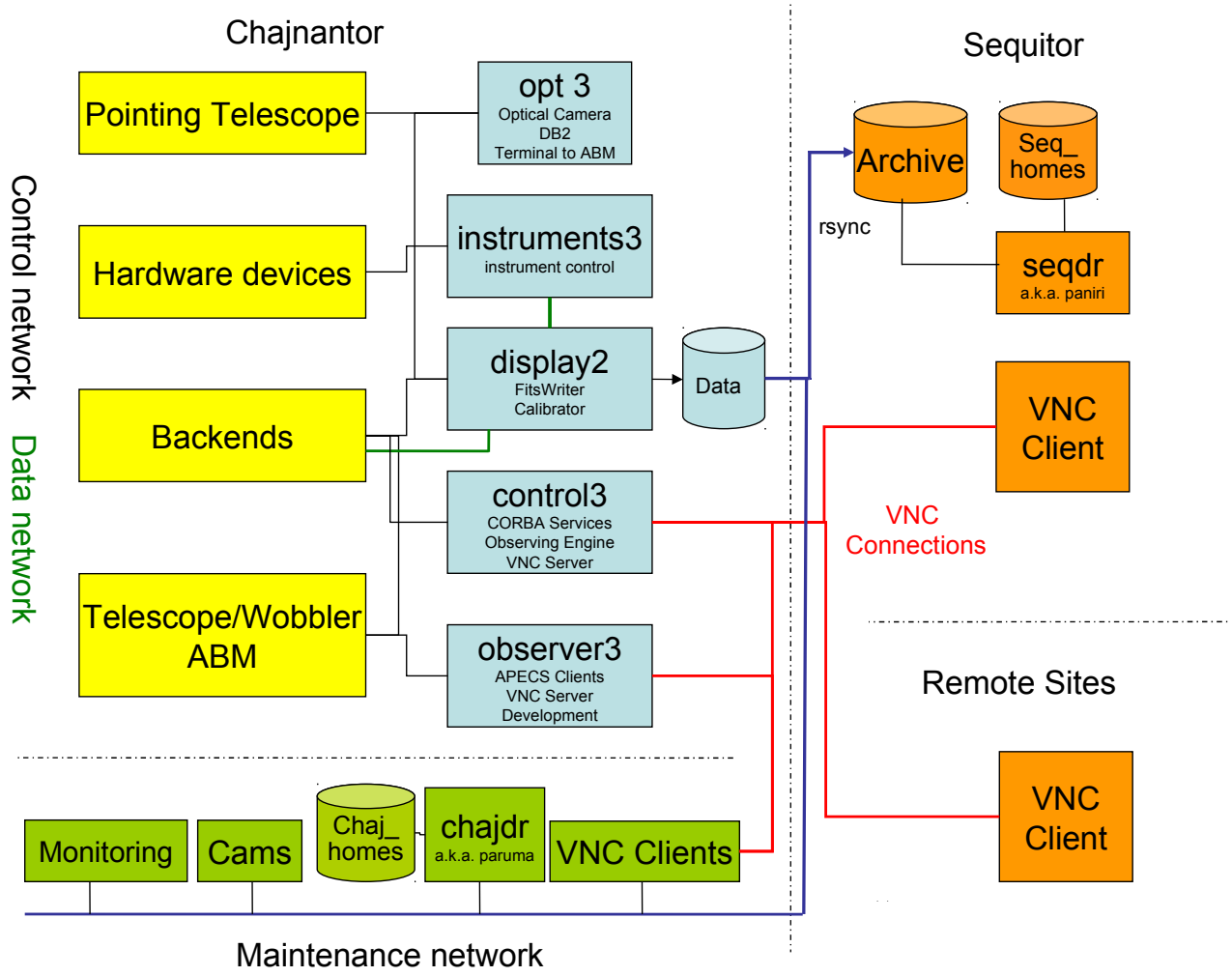


Figure 4.1: *APECS deployment at the site in Chile.*

APECS. One thus needs to increase the tracking tolerance to about 5–10 arcseconds using the `tolerance` command in `apecs`.
After a couple of minutes the system will be up and running and a number of windows and displays will open. Table 4.1 summarizes the processes that need to be running for APECS to work. The troubleshooting section refers to the commands in this table when suggesting to restart part of the system.

## 4.3 Configurations

APECS uses a number of configurations from the area defined by the `$APECSCONFIG` environment variable:

- `$APECSCONFIG/Pointing`: Pointing models.
- `$APECSCONFIG/Focus`: Focus models.
- `$APECSCONFIG/FeedParams`: Detailed frontend feed configurations.
- `$APECSCONFIG/RCPs`: Frontend feed offsets and gains.
- `$APECSCONFIG/Catalogs`: System source and line catalogs.
- `$APECSCONFIG/Ephemerides`: Ephemeris files.

The pointing models are split into a master model (`MASTER.ptg`) which describes the overall antenna behavior and delta models for frontends (see below), tiltmeter corrections (`TILT.ptg`) and linear sensor

corrections (`DISP.ptg`). The master model is currently based on the optical pointing model and might be replaced later by bolometer radio pointing models.
The receiver delta pointing models are named after their `apecs` name, e.g. `HET345.ptg`. They contain only the deltas to be added to the master model. For Cassegrain receivers there are usually just two offset numbers, for Nasmyth receivers there are 4 constants.
The ephemeris files are stored in `xephem` "`edb`" format. Ephemerides for comets and distant objects are installed.
The scan number is being increased using the global scan number in `$APECSCONFIG/globalScanProperties`. The scan number is automatically reset to "1" on January 1$^{st}$ every year.
Some of the non-configurable instrument setup that is not included in `$APECSCONFIG/FeedParams` is still hardcoded in the `apexFEBEConfig.py` Python module which is located in the `$APECSROOT/lib/python/site-packages` directory. It is assumed that this stays stable once the instrument has been commissioned and accepted. Changing this setup is only possible under version control by a software responsible.
Note that the ethernet switch port used to connect the "ABM" VME computer needs to be configured to "100half" with "auto-negotiation" disabled. The MVME 761 board cannot handle "auto-negotiation".

## 4.4 Troubleshooting

While APECS is now rather stable compared to the initial phases of the APEX commissioning, there are still some situations that need human intervention. First and foremost always check the messages in the "`jlog`" if an error occurs. Most conditions can be diagnosed by evaluating those messages. All logging messages are stored in the "`$APECSSYSLOGS`" directory. One will also find there all the details about the SCPI communication to the embedded systems.
For a proper debugging, one needs to have at least a rough idea of how the system works. The details of the APECS design have been compiled in some training documents. Here we give just a short summary of the setup. The APECS system is using CORBA as the main communication mechanism. A number of CORBA services (naming, notification, CDB etc.) are started first. Without those services, the system will not work at all.
There is a central manager process that knows about the layout of the system, in particular which process is running on which machine. Next there are a number of so called container processes in which the CORBA objects are being instantiated. The CORBA objects are mostly used to map hardware functionality into the control system.
For APEX it was decided to not burden the instrument developers with CORBA software development which can be quite heavy. Instead we agreed on a simple text protocol based on the SCPI standard to send commands to the instrument control systems and to receiver monitor point information.
Thus the APEX CORBA objects are simply a wrapper between CORBA requests and corresponding SCPI commands sent via UDP socket connections to the embedded systems ([1]). They are therefore automatically generated from the IDL interface files. The ABM CORBA objects inherited from ALMA via the TICS software are different. They are implemented using the CAN bus communication directly without additional layers.
It is important to understand this layered structure of the APECS system. The top-level communication errors that the user sees can have different causes either on the CORBA container level or on the SCPI level. We currently can not yet route the nature of the error all the way up to the user. This may change in the future but in the current setup one needs to examine all communication layers to debug the problems.
The collection of CORBA objects representing the hardware is being coordinated by the central "Observing Engine" process. It sets up all instruments, auxiliary devices and the antenna itself for each scan. If one of those setups fails, the whole scan is canceled. This is usually the point where observers turn to operators or software engineers for help. You then need to examine the system and the log messages to identify the offending sub-system.

| Host | Applications | Process / Command |
|---|---|---|
| control3 | ACS CORBA Services | `acsStart` |
| | CAN Monitor | `canMonitor` |
| | Observation Logger Server | `apexObsLoggerServer` |
| | Observing Engine | `apexObsEngine` |
| instruments3 (APECS) | `frontends` Container | `runContainer frontends` |
| | `backends` Container | `runContainer backends` |
| | `aux` Container | `runContainer aux` |
| | `infra` Container | `runContainer infra` |
| | `environ` Container | `runContainer environ` |
| | `opttel` Container | `runContainer opttel` |
| | Monitor point archiver | `apexMonPointArchiver` / `MySQL` |
| instruments3 (Embedded System Servers included in APECS) | Weather station SCPI CID Server | `apexWeatherStation start` |
| | Radiometer SCPI CID Server | `apexRadiometer start` |
| | Tiltmeter SCPI CID Server | `apexTiltmeter start` |
| | TSGen SCPI CID Server | `apexTSGen start` |
| | PBE_A SCPI CID Server | `apexPBE_A start` |
| | PBE_B SCPI CID Server | `apexPBE_Bholo start` |
| | PBE_C SCPI CID Server | `apexPBE_C start` |
| | PBE_D SCPI CID Server | `apexPBE_D start` |
| | PBE_E SCPI CID Server | `apexPBE_Eac start` |
| | PBE_F SCPI CID Server | `apexPBE_F start` |
| | FLASH emulator | `emuEmbSys APEX_FLASH 0` |
| | CALUNITS:B emulator | `emuEmbSys APEX_CALUNITS_B 0` |
| | CALUNITS:C emulator | `emuEmbSys APEX_CALUNITS_C 0` |
| display2 | Online MBFITS Writer | `apexOnlineFitsWriter` |
| | Online Calibrator | `apexOnlineCalibrator` |
| | Cal. Display Server | `apexCalibDisplayServer` |
| opt3 | `opt` Container | `runContainer opt` |
| abm | `abm1` Container | `apexLCUContainerStart --container=abm1 \ --netname=abm # (on control3)` |
| observer3 | Observing CLI | `apecs` |
| | Scan summary status | `scanStatus` |
| | APECS logging messages | `apecsLog` |
| | Antenna position | `apexAntMount` |
| | Sun avoidance display | `plotAvoidance` |
| | PWV history display | `apexPWVHistory` |
| | Observation Logger Client | `apexObsLoggerClient` |
| | Online Pipeline Display | `apexCalibratorClient` |
| | Shutter Control GUI | `shutterControl` |
| observer3 | Main monitoring client | `apexStatusDisplay \ masterStatus.MasterStatus` |
| | Weather display | `apexWeatherDisplay` |
| | Low-level ACU/WIU access | `acu_wiu` |
| paruma / paniri | Data reduction software | `class` / `boa` / `crush` |
| apexmysql | Project SQL database | mysqld |

Table 4.1: This table shows the APECS hosts, typical processes and their shell commands. The standard server processes on `control3`, `instruments3`, `display2`, `opt3` and `abm` are started by `restartAPECSServers` (on `control3`). The server processes must run under the `apex` operations account. The recommended observing setup comprises the observer client processes on `observer3` as started by `restartAPECSObsClients` and the monitoring client processes as started using `restartAPECSMonClients`. All client processes must be started under the observing project account in order to get access to macros, catalogs, raw and scientific data, and observing logs.

Table 4.2 summarizes a number of known conditions and their usual solution. Sometimes the suggested measures might not resolve the problem and further debugging is needed. Always try to get the system back into operation by restarting only small portions of it rather than simply restarting everything. This way one learns more about the structure of the system and it is faster too.

| Symptom | Cause | Solution |
|---|---|---|
| The scan gets stuck at the end of a subscan. | The online calibrator does not return from the CORBA "reduce" command and thus stalls the Observing Engine. | Restart the online calibrator by typing `onlineCalibrator restart` in a terminal in the `control3:1` VNC session. |
| The telescope can not be taken into track mode when starting a scan. | This can happen if some telescope interlocks have been activated (e.g. by lowering the platform to the Cassegrain cabin). | Clear all ACU errors (`acu.clear_fault_cmd()`), press the "standby" button on the `apexAntMount` GUI, and try the scan again. |
| There is no data in one or more of the MBFITS ARRAYDATA tables. | The TCP connection to one or more backends may have failed. | Check the backend control programs. Check the socket status on port 25144 on `display2` and on the backend host. If there are `CLOSE_WAIT` conditions, you may need to wait 10 minutes for Linux to release the resource. In the worst case the backend computers may need to be restarted. |
| Setting or reading instrument parameters fails and scans are canceled. | Most likely the embedded system control software has failed. Very rarely the CORBA containers may crash. | There is an ACS timeout period of 5 seconds when accessing properties. If a `getMCPoint` returns faster than this and fails, then the CORBA object is gone and the container may have crashed. Restart the container. Otherwise the socket communication has timed out (4 seconds). In this case check the corresponding SCPI CID server processes on the APECS machines (see table 4.1) or on the embedded systems and try restarting the server processes. If this does not help, log on to `instruments3` and check the containers as described above. |
| The scan gets stuck, LST is not updating anymore in the scan status, the MBFITS writer reports increasing monitor point delays of many seconds. | The antenna CORBA components are no longer available. | Restart APECS. |
| Restarting the APECS servers does not start the ABM properly. | The ABM may have lost its ethernet connection and can no longer be booted remotely. | Type <CTRL-X> in the `startMiniEtherABM` window on `control3` to reboot the ABM. Wait until it is back up and then restart APECS again. |

Table 4.2: Known APECS problems and their solutions.

With APECS 2.0 we began providing a number of individual restart scripts for the core applications. This helps in restarting only portions of the system and it also makes sure that the application is running under the correct account. The restart scripts need to be run as user `apex` on `control3`. Table 4.3 shows the available commands.

| Application | Restart command |
|---|---|
| Online Calibrator | `onlineCalibrator start \| stop \| restart` |
| MBFITS Writer | `fitsWriter start \| stop \| restart` |
| Observing Logger Server | `obsLoggerServer start \| stop \| restart` |
| Raw Data Copier | `rawdataCopier start \| stop \| restart` |
| Observing Engine | `obsEngine start \| stop \| restart` |

Table 4.3: Core APECS application restart commands.

## 4.5 Notes and Caveats

– Note that using only `stopAPECSServers` followed by `startAPECSServers` will **not** stop any clients. The idea is to leave the clients running so that they can re-connect once the servers are back up. This only partially works in APECS 3.1. We still recommend to always use `restartAPECSServers` or issue an explicit `stopAllAPECSClients` before stopping the servers.

– The `abm` computer's console is connected to the ethernet via an RS232 line and a COM server which is accessible through the `startMiniEtherABM` command on `control3`. One can thus access the `abm` in a way that is independent of its normal ethernet connection to check its status or possibly reboot it by typing `reboot` at the `abm1->` prompt or, if it is not reacting anymore, by typing <CTRL-X>.

– Note that the `startMiniEtherABM` job must not be terminated as this leads to filling the RS232 output buffer which eventually gets the `abm` stuck and then requires its reboot.

– **Never** type `exit` at the `abm` prompt as this closes the login shell and the computer needs to be rebooted. Use `logout` if you are connected via `rlogin` or `rsh`.

# Chapter 5

# CORBA CDB Reference

Tables 5.1 to 5.6 summarize the names of all APECS CORBA objects as defined in the CDB (Configuration Database). The names are needed to access the objects e.g. via the `apexObsUtils.getMCPoint` command.

| CORBA Object Name | Container | Embedded System Host(s) | Process |
|---|---|---|---|
| ABM0/CLOCK | abm0 | control3 | maciContainer abm0 |
| ABM0/TIMER | abm0 | control3 | maciContainer abm0 |
| ABM0/TRAJECTORY | abm0 | control3 | maciContainer abm0 |
| ABM0/ANTMOUNT | abm0 | control3 | maciContainer abm0 |
| ABM0/WOBBLER | abm0 | control3 | maciContainer abm0 |
| ABM1/CLOCK | abm1 | abm | maciContainer abm1 |
| ABM1/TIMER | abm1 | abm | maciContainer abm1 |
| ABM1/AMBCAN | abm1 | abm | maciContainer abm1 |
| ABM1/TRAJECTORY | abm1 | acu | maciContainer abm1 |
| ABM1/ANTMOUNT | abm1 | acu | maciContainer abm1 |
| ABM1/WOBBLER | abm1 | wiu | maciContainer abm1 |
| ABM1/ACU | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/METROLOGY | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/METROLOGY/TILTMETERS | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/METROLOGY/TEMPERATURES | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/METROLOGY/COEFF | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/METROLOGY/DELTAS | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/SUBREF | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/SUBREF/HEXAPOD | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/UPS/UNIT1 | aux | acu-cpu2 | canMonitor |
| ABM1/ACU/UPS/UNIT2 | aux | acu-cpu2 | canMonitor |
| OPT/CLOCK | opt | opt3 | maciContainer opt |
| OPT/TIMER | opt | opt3 | maciContainer opt |
| OPT/OPTPOINT | opt | opt3 | maciContainer opt |
| APEX/CALUNITS/A | aux | cs-a1 | calunit |
| APEX/CALUNITS/A/MIRRORS | aux | cs-a1 | calunit |
| APEX/CALUNITS/A/COOLER | aux | cs-a1 | calunit |
| APEX/CALUNITS/B | aux | instruments3 | emuEmbSys |
| APEX/CALUNITS/C | aux | instruments3 | emuEmbSys |

Table 5.1: APECS CORBA components, containers, server hosts and processes providing the actual implementation.

| CORBA Object Name | Container | Embedded System Host(s) | Process |
|---|---|---|---|
| APEX/HOLO | frontends | instruments3 | emuEmbSys |
| APEX/LABOCA | frontends | laboca | N/A |
| APEX/LABOCA/CALUNIT | frontends | laboca | N/A |
| APEX/LABOCA/AMPLIFIER | frontends | laboca | N/A |
| APEX/LABOCA/POLARIMETER | frontends | laboca | N/A |
| APEX/LABOCA/MAINTENANCE | frontends | laboca | N/A |
| APEX/SABOCA | frontends | laboca | N/A |
| APEX/SABOCA/AMPLIFIER | frontends | laboca | N/A |
| APEX/BOLOSZ | frontends_old | bolosz | N/A |
| APEX/BOLOSZ/FRIDGE | frontends_old | bolosz | N/A |
| APEX/BOLOSZ/MAINTENANCE | frontends_old | bolosz | N/A |
| APEX/AMKID350 | frontends | a-mkid-fe | N/A |
| APEX/AMKID870 | frontends | a-mkid-fe | N/A |
| APEX/AMKID/CALUNIT | frontends | a-mkid-fe | N/A |
| APEX/ARTEMIS200 | frontends | emuEmbSys | N/A |
| APEX/ARTEMIS350 | frontends | emuEmbSys | N/A |
| APEX/ARTEMIS450 | frontends | emuEmbSys | N/A |
| APEX/ARTEMIS/OPTICS | aux | cs-a1 | N/A |
| APEX/BOLOB | frontends_old | instruments3 | emuEmbSys |
| APEX/HET230 | frontends | hfe-control | N/A |
| APEX/HET230/LO1 | frontends | hfe-control | N/A |
| APEX/HET345 | frontends | hfe-control | N/A |
| APEX/HET345/LO1 | frontends | hfe-control | N/A |
| APEX/HET460 | frontends | hfe-control | N/A |
| APEX/HET460/LO1 | frontends | hfe-control | N/A |
| APEX/HET1300 | frontends | hfe-control | N/A |
| APEX/HET1300/LO1 | frontends | hfe-control | N/A |
| APEX/FLASH345 | frontends_new | flash | LabView |
| APEX/FLASH345/LO1 | frontends_new | flash | LabView |
| APEX/FLASH460L | frontends_new | flash | LabView |
| APEX/FLASH460L/LO1 | frontends_new | flash | LabView |
| APEX/FLASH460H | frontends_new | flash | LabView |
| APEX/FLASH460H/LO1 | frontends_new | flash | LabView |
| APEX/CHAMP690 | frontends | champ | N/A |
| APEX/CHAMP810 | frontends | champ | N/A |
| APEX/CHAMP/DEROTATOR | frontends | champ | N/A |
| APEX/CHAMP/CALUNIT | frontends | champ | N/A |
| APEX/PI460 | frontends | instruments3 | emuEmbSys |
| APEX/PI1100 | frontends | pi1100 | N/A |
| APEX/ZSPEC | frontends_old | instruments3 | emuEmbSys |
| APEX/ZEUS2 | frontends_old | instruments3 | emuEmbSys |
| APEX/SUPERCAM | frontends_pi | supercam | N/A |
| APEX/SUPERCAM/CALUNIT | frontends_pi | supercam | N/A |
| APEX/GARD180 | frontends | gard180 | N/A |
| APEX/GARD180/LO1 | frontends | gard180 | N/A |
| APEX/GARD180/CALUNIT | frontends | gard180 | N/A |

Table 5.2: APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.).

| CORBA Object Name | Container | Embedded System Host(s) | Process |
|---|---|---|---|
| APEX/ABBA | backends | abba | N/A |
| APEX/PBE_A | backends | instruments3 | apexPBEControl |
| APEX/PBE_B | backends | instruments3 | apexPBEControl |
| APEX/PBE_C | backends | instruments3 | apexPBEControl |
| APEX/PBE_D | backends | instruments3 | apexPBEControl |
| APEX/PBE_E | backends | instruments3 | apexPBEControl |
| APEX/PBE_F | backends | instruments3 | apexPBEControl |
| APEX/SZACBE | backends_old | szbackend | N/A |
| APEX/SZDCBE | backends_old | szbackend | N/A |
| APEX/BEAR1 | backends | bear1 | N/A |
| APEX/BEAR2 | backends | bear2 | N/A |
| APEX/AMKID350BE | backends | a-mkid-fe | N/A |
| APEX/AMKID870BE | backends | a-mkid-fe | N/A |
| APEX/IF1 | aux | if_control / if_proc1 | N/A |
| APEX/IF1/CHAIN1 | aux | if_control / if_proc1 | N/A |
| APEX/IF1/CHAIN2 | aux | if_control / if_proc1 | N/A |
| APEX/IF2 | aux | if_control / if_proc2 | N/A |
| APEX/IF2/CHAIN1 | aux | if_control / if_proc2 | N/A |
| APEX/IF2/CHAIN2 | aux | if_control / if_proc2 | N/A |
| APEX/IF3 | aux | if3 | N/A |
| APEX/IF3/CHAIN1 | aux | if3 | N/A |
| APEX/IF3/CHAIN2 | aux | if3 | N/A |
| APEX/IF3/CHAIN3 | aux | if3 | N/A |
| APEX/IF3/CHAIN4 | aux | if3 | N/A |
| APEX/IF4 | aux | if4 | N/A |
| APEX/IF4/CHAIN1 | aux | if4 | N/A |
| APEX/IF4/CHAIN2 | aux | if4 | N/A |
| APEX/IF5 | aux | if5 | N/A |
| APEX/IF5/CHAIN1 | aux | if5 | N/A |
| APEX/IF5/CHAIN2 | aux | if5 | N/A |
| APEX/IF5/CHAIN3 | aux | if5 | N/A |
| APEX/IF5/CHAIN4 | aux | if5 | N/A |
| APEX/IFC1 | aux | if_control / champ | N/A |
| APEX/IFC1/CHAIN1 | aux | if_control / champ | N/A |
| APEX/IFC1/CHAIN2 | aux | if_control / champ | N/A |
| APEX/IFC2 | aux | if_control / champ | N/A |
| APEX/IFC2/CHAIN1 | aux | if_control / champ | N/A |
| APEX/IFC2/CHAIN2 | aux | if_control / champ | N/A |
| APEX/IFV1 | aux | instruments3 | apexVLBIDownConverterControl |
| APEX/IFV1/CHAIN1 | aux | instruments3 | apexVLBIDownConverterControl |
| APEX/IFV1/CHAIN2 | aux | instruments3 | apexVLBIDownConverterControl |

Table 5.3: APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.).

| CORBA Object Name | Container | Embedded System Host(s) | Process |
|---|---|---|---|
| APEX/AFFTS | backends | affts | N/A |
| APEX/AFFTS/BAND1 | backends | affts | N/A |
| APEX/AFFTS/BAND2 | backends | affts | N/A |
| APEX/AFFTS/BAND3 | backends | affts | N/A |
| APEX/AFFTS/BAND4 | backends | affts | N/A |
| APEX/AFFTS/BAND5 | backends | affts | N/A |
| APEX/AFFTS/BAND6 | backends | affts | N/A |
| APEX/AFFTS/BAND7 | backends | affts | N/A |
| APEX/AFFTS/BAND8 | backends | affts | N/A |
| APEX/AFFTS/BAND9 | backends | affts | N/A |
| APEX/AFFTS/BAND10 | backends | affts | N/A |
| APEX/AFFTS/BAND11 | backends | affts | N/A |
| APEX/AFFTS/BAND12 | backends | affts | N/A |
| APEX/AFFTS/BAND13 | backends | affts | N/A |
| APEX/AFFTS/BAND14 | backends | affts | N/A |
| APEX/AFFTS/BAND15 | backends | affts | N/A |
| APEX/AFFTS/BAND16 | backends | affts | N/A |
| APEX/AFFTS/BAND17 | backends | affts | N/A |
| APEX/AFFTS/BAND18 | backends | affts | N/A |
| APEX/AFFTS/BAND19 | backends | affts | N/A |
| APEX/AFFTS/BAND20 | backends | affts | N/A |
| APEX/AFFTS/BAND21 | backends | affts | N/A |
| APEX/AFFTS/BAND22 | backends | affts | N/A |
| APEX/AFFTS/BAND23 | backends | affts | N/A |
| APEX/AFFTS/BAND24 | backends | affts | N/A |
| APEX/AFFTS/BAND25 | backends | affts | N/A |
| APEX/AFFTS/BAND26 | backends | affts | N/A |
| APEX/AFFTS/BAND27 | backends | affts | N/A |
| APEX/AFFTS/BAND28 | backends | affts | N/A |
| APEX/AFFTS/BAND29 | backends | affts | N/A |
| APEX/AFFTS/BAND30 | backends | affts | N/A |
| APEX/AFFTS/BAND31 | backends | affts | N/A |
| APEX/AFFTS/BAND32 | backends | affts | N/A |
| APEX/XFFTS | backends | xffts | N/A |
| APEX/XFFTS/BAND1 | backends | xffts | N/A |
| APEX/XFFTS/BAND2 | backends | xffts | N/A |
| APEX/XFFTS/BAND3 | backends | xffts | N/A |
| APEX/XFFTS/BAND4 | backends | xffts | N/A |
| APEX/XFFTS/BAND5 | backends | xffts | N/A |
| APEX/XFFTS/BAND6 | backends | xffts | N/A |
| APEX/XFFTS/BAND7 | backends | xffts | N/A |
| APEX/XFFTS/BAND8 | backends | xffts | N/A |
| APEX/XFFTS2 | backends | xffts2 | N/A |
| APEX/XFFTS2/BAND1 | backends | xffts2 | N/A |
| APEX/XFFTS2/BAND2 | backends | xffts2 | N/A |
| APEX/ZSPECBE | backends_old | zspecbe | N/A |
| APEX/ZSPECBE/BAND1 | backends_old | zspecbe | N/A |
| APEX/ZSPECBE/BAND2 | backends_old | zspecbe | N/A |
| APEX/ZSPECBE/BAND3 | backends_old | zspecbe | N/A |
| APEX/ZEUS2BE | backends_old | zeus2be | N/A |

Table 5.4: APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.).

| CORBA Object Name | Container | Embedded System Host(s) | Process |
|---|---|---|---|
| APEX/SCBE | backends_pi | scbe | N/A |
| APEX/SCBE/BAND1 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND2 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND3 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND4 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND5 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND6 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND7 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND8 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND9 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND10 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND11 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND12 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND13 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND14 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND15 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND16 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND17 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND18 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND19 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND20 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND21 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND22 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND23 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND24 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND25 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND26 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND27 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND28 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND29 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND30 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND31 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND32 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND33 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND34 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND35 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND36 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND37 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND38 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND39 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND40 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND41 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND42 | backends_pi | scbe | N/A |
| ... | ... | ... | ... |
| APEX/SCBE/BAND60 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND61 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND62 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND63 | backends_pi | scbe | N/A |
| APEX/SCBE/BAND64 | backends_pi | scbe | N/A |

Table 5.5: APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.).

| CORBA Object Name | Container | Embedded System Host(s) | Process |
|---|---|---|---|
| APEX/OPTICS/A | aux | cs-a1 | apexOpticsAControl |
| APEX/OPTICS/B | aux | optics-b | N/A |
| APEX/OPTICS/C | aux | cs-a1 | cmirror |
| APEX/OPTTEL | opttel | instruments3 | N/A |
| APEX/SYNTHESIZER1 | aux | apexcool | compdata & apexSynthesizerControl |
| APEX/SYNTHESIZER1/DESTINATION | aux | apexcool | compdata apexSynthesizerControl |
| APEX/SYNTHESIZER1/FSUNIT | aux | apexcool | compdata apexSynthesizerControl |
| APEX/SYNTHESIZER2 | aux | apexcool | compdata apexSynthesizerControl |
| APEX/SYNTHESIZER2/DESTINATION | aux | apexcool | compdata apexSynthesizerControl |
| APEX/SYNTHESIZER2/FSUNIT | aux | apexcool | compdata apexSynthesizerControl |
| APEX/REFPLLSYNTHESIZER | aux | apexcool | compdata apexSynthesizerControl |
| APEX/REFILLTANK | infra | apexcool | compdata & pt100server |
| APEX/TEMPERATURES | infra | apexcool | pt100server |
| APEX/TSGEN | aux | instruments3 | apexTSGenControl |
| APEX/WEATHERSTATION | environ | instruments3 | apexWeatherStationControl |
| APEX/RADIOMETER | environ | instruments3 | apexRadiometerControl |
| APEX/RADIOMETER/RESULTS | environ | instruments3 | apexRadiometerResultsServer |
| APEX/RADIOMETER/MAINTENANCE | environ | instruments3 | apexRadiometerControl |
| APEX/TILTMETERS/BASE | infra | instruments3 | apexTiltmeterControl |
| APEX/RACKCHILLER | infra | apexcool | compdata & pt100server |
| APEX/COMPRESSORCHILLER | infra | apexcool | compdata & pt100server |
| APEX/PCCHILLER | infra | apexcool | compdata & pt100server |
| APEX/COMPRESSOR1 | infra | apexcool | compdata & pt100server |
| APEX/COMPRESSOR2 | infra | apexcool | compdata & pt100server |
| APEX/COMPRESSOR3 | infra | apexcool | compdata & pt100server |
| APEX/COMPRESSOR4 | infra | apexcool | compdata & pt100server |
| APEX/DOORS | infra | apexcool | compdata & pt100server |

Table 5.6: APECS CORBA components, containers, server hosts and processes providing the actual implementation (ctd.).

# Bibliography

[1] Hafok, H., Muders, D. & Olberg, M., 2006, APEX SCPI Socket Command Syntax and Backend Data Stream Format, APEX Report APEX-MPI-ICD-0005, Rev. 1.0

[2] Muders, D., 2007, APEX Instruments Generic CORBA IDL Interfaces, APEX Report APEX-MPI-ICD-0004, Rev. 1.9

[3] Muders, D., Hafok, H., Wyrowski, F., Polehampton, E., Belloche, A., König, C., Schaaf, R., Schuller, F., Hatchell, J., v.d.Tak, F., 2006, APECS - The Atacama Pathfinder Experiment Control System, *A&A* Letters, 454, L25-L28

[4] Muders, D., Hatchell, J., Lemke, R., Olberg, M. & Hafok, H., 2002, Software Interfaces for Submillimeter Telescope Instrumentation, APEX Report APEX-ICD-MPI-0001

[5] Muders, D., Polehampton, E. & Hatchell, J., 2007, Multi-beam FITS Raw Data Format, APEX Report APEX-MPI-ICD-0002, Rev. 1.62

[6] Pardo, J. R., Cernicharo, J. & Serabyn, 2001, Atmospheric Transmission at Microwaves (ATM): An Improved Model for Millimeter/Submillimeter Applications, IEEE Trans. on Antennas and Propagation, 49, 1683

[7] Polehampton, E., 2005, APEX Calibrator Manual, APEX Report APEX-MPI-MAN-0012

[8] Rossum, G. v., Drake Jr., F.L., 2010, Python Reference Manual, Release 2.6.5, http://docs.python.org/release/2.6.5